2025 - 2026

Loan THOMY

Nexa Digital School

Étude de cas - Bloc 1

1. Reformuler les besoins métiers et technique de LogiNord

1.1 - Contexte global

LogiNord est une entreprise de logistique e-commerce en forte croissance implémentée dans les Hauts-de-France. Elle souhaite mettre en place une solution d'intelligence artificielle afin de mieux planifier la charge et optimiser l'allocation des ressources. Elle rencontre des difficultés liées à l'hétérogénéité de ses sources, à la qualité des données et à des temps de traitement trop élevés.

1.2 - Problématiques

- Données dispersées dans plusieurs formats / systèmes : une consolidation est nécessaire
- Volumétrie et croissance : nécessité d'évaluer la faisabilité technique, le choix du SGBD (SQL ou NoSQL)
- Qualité des données (valeurs manquantes, incohérences) : détection, correction et compatibilité avec une base
- Contraintes réglementaires et éthiques, RGPD, AI Act, confidentialité et bonnes pratiques à intégrer dès le début de la collecte des données.

1.2 - Objectifs à atteindre

1. Analyser et reformuler le besoin, proposer une stratégie technique adaptée
2. Auditer / cartographier les données et vérifier leur exploitabilité
3. Choisir et installer un SGBD adapté à la volumétrie et vérifier son bon fonctionnement
4. Agréger des fichiers hétérogènes, harmoniser puis traiter les valeurs manquantes selon le type (qualitatif ou quantitatif)
5. Charger des données nettoyées en garantissant l'intégrité
6. Optimiser la base de donnée et comparer les temps d'exécutions entre tables brutes et optimisées
7. Document RGPD et éthique

1.3 Distinction des besoins métiers et des besoins techniques

Besoins **métiers** :

- Une vue consolidée commandes / clients / entrepôts pour piloter le flux
- Des indicateurs fiables (charge, volume, anomalies) pour décider rapidement
- Une base de données et des traitements qui tiennent la montée en charge (performance)
- Conformité RGPD et cadre éthique (pas d'usage abusif des données clients)

Besoins **techniques** :

- Cartographie des données et validation d'exploitabilité
- Choix + installation d'un SGBD compatible Windows / Linux, vérification via tables par défaut

- Pipeline d'ingestion multi-formats avec harmonisation des schémas, types, formats de dates, normalisation des référentiels (exemple : entrepôts)
- Règles de traitement des valeurs manquantes et traçabilité
- Chargement contrôlé, comptage avant / après + contrôle de lisibilité des types
- Optimisation : types adaptés + taille + indexation puis benchmark de requêtes

2. Tableau de synthèse des sources

Source	Format	Description	Variables (exemples)	Typologie	Volumétrie (estimation)
commandes_lognord	CSV	Flux de commandes e-commerce	order_id, client_id, date	Structurées, quantitatives + qualitatives	Environ 3 000 à 8 000 lignes / jour
référetiel_clients	Excel (XLSX)	Référentiel client	client_id, code_postal,	Structurées, qualitatives + données perso	Environ le nombre de clients actifs
référentiels_entrepots	JSON	Métadonnées entrepôts	warehouse_name,	Semi-structurées (JOSN), qualitatives + quantitatives	Faible (dizaine / centaine d'entrepôts)
logs_anomalies	JSON / CSV	Liste d'erreurs	order_id	Semi-structurées	Variable car c'est liée au taux d'erreur

3. Choix et installation d'un SGBD

SGBD choisi : PostgreSQL (SQL)

- Le besoin vise consolidation "commandes / clients / entrepôts" : modèle relationnel naturel (clés, intégrité, jointures)
- Selon la volumétrie et la mesure de la performance des requêtes, PostgreSQL est idéal, notamment pour l'indexation, EXPLAIN ANALYSE et l'optimisation des types
- PostgreSQL supporte également le JSON / JSONB si cela est nécessaire tout en restant en SQL

Etant sur Fedora 43 (Linux), voici les commandes que j'ai fais pour effectuer l'installation :

3.1 - Mise à jour du système + installation de PostgreSQL

sudo dnf update

sudo dnf install postgresql-server postgresql-contrib

3.2 - Initialisation de la base PostgreSQL

sudo postgresql-setup –initdb

```
loan@Loan-fedora:/mnt/data/Documents/Doranco/Partiels/Février$ sudo postgresql-setup --initdb
 * Initializing database in '/var/lib/pgsql/data'
 * Initialized, logs are in /var/lib/pgsql/initdb_postgresql.log
```

3.3 - Démarrage et activation du service

sudo systemctl enable –now postgresql

sudo systemctl status postgresql

```
loan@Loan-fedora:/mnt/data/Documents/Doranco/Partiels/Février$ sudo systemctl enable --now postgresql
Created symlink '/etc/systemd/system/multi-user.target.wants/postgresql.service' → '/usr/lib/systemd/system/postgresql.service'.
loan@Loan-fedora:/mnt/data/Documents/Doranco/Partiels/Février$ sudo systemctl status postgresql
● postgresql.service - PostgreSQL database server
     Loaded: loaded (/usr/lib/systemd/system/postgresql.service; enabled; preset: disabled)
    Drop-In: /usr/lib/systemd/system/service.d
             └─10-timeout-abort.conf
     Active: active (running) since Mon 2026-02-02 11:22:01 CET; 11s ago
 Invocation: 5da6c29323864470b5034934731e46bd
    Process: 54510 ExecStartPre=/usr/libexec/postgresql-check-db-dir postgresql (code=exited, status=0/SUCCESS)
   Main PID: 54512 (postgres)
      Tasks: 10 (limit: 9144)
     Memory: 34.2M (peak: 35.3M)
        CPU: 41ms
     CGroup: /system.slice/postgresql.service
             ├─54512 /usr/bin/postgres -D /var/lib/pgsql/data
             ├─54514 "postgres: logger "
             ├─54515 "postgres: io worker 0"
             ├─54516 "postgres: io worker 1"
             ├─54517 "postgres: io worker 2"
             ├─54518 "postgres: checkpointer "
             ├─54519 "postgres: background writer "
             ├─54521 "postgres: walwriter "
             ├─54522 "postgres: autovacuum launcher "
             └─54523 "postgres: logical replication launcher "

févr. 02 11:22:01 Loan-fedora systemd[1]: Starting postgresql.service - PostgreSQL database server...
févr. 02 11:22:01 Loan-fedora postgres[54512]: 2026-02-02 11:22:01.398 CET [54512] LOG:  redirection des traces vers le processus de récupération des traces
févr. 02 11:22:01 Loan-fedora postgres[54512]: 2026-02-02 11:22:01.398 CET [54512] ASTUCE :  Les prochaines traces apparaîtront dans le répertoire « log ».
févr. 02 11:22:01 Loan-fedora systemd[1]: Started postgresql.service - PostgreSQL database server.
```

3.4 - Accès au shell PostgreSQL

sudo -u postgre psql

```
loan@Loan-fedora:/mnt/data/Documents/Doranco/Partiels/Février$ sudo -u postgres psql
psql (18.1)
Saisissez « help » pour l'aide.

postgres=# \l
                                                  Liste des bases de données
    Nom    | Propriétaire | Encodage | Fournisseur de locale | Collationnement | Type caract. | Locale | Règles ICU : |    Droits d'accès
-----------+--------------+----------+-----------------------+-----------------+--------------+--------+--------------+-----------------------
 postgres  | postgres     | UTF8     | libc                  | fr_FR.UTF-8     | fr_FR.UTF-8  |        |              |
 template0 | postgres     | UTF8     | libc                  | fr_FR.UTF-8     | fr_FR.UTF-8  |        |              | =c/postgres          +
           |              |          |                       |                 |              |        |              | postgres=CTc/postgres
 template1 | postgres     | UTF8     | libc                  | fr_FR.UTF-8     | fr_FR.UTF-8  |        |              | =c/postgres          +
           |              |          |                       |                 |              |        |              | postgres=CTc/postgres
(3 lignes)
```

3.5 - Création de la base de donnée

```
postgres=# CREATE DATABASE loginord;
CREATE DATABASE
postgres=# \c loginord
Vous êtes maintenant connecté à la base de données « loginord » en tant qu'utilisateur « postgres ».
```

3.6 - Création des tables brutes

```
loginord=# CREATE TABLE stg_orders_raw(order_id TEXT, client_id TEXT, order_date TEXT, amount TEXT, warehouse TEXT);
CREATE TABLE stg_clients_raw(client_id TEXT, city TEXT, postal_code TEXT, segment TEXT);
CREATE TABLE
CREATE TABLE
loginord=# CREATE TABLE stg_warehouses_raw (warehouse_code TEXT, warehouse_name TEXT, region TEXT, capacity TEXT);
CREATE TABLE
```

3.7 - Chargement des fichiers dans les tables brutes

```
loginord=# CREATE TABLE stg_warehouses_raw (warehouse_code TEXT, warehouse_name TEXT, region TEXT, capacity TEXT);
CREATE TABLE
loginord=# \copy stg_orders_raw FROM '/mnt/data/Documents/Doranco/Partiels/Février/commandes_lognord.csv' DELIMITER ',' CSV HEADER;
COPY 5
loginord=# \copy stg_clients_raw FROM '/mnt/data/Documents/Doranco/Partiels/Février/clients_loginord.csv' DELIMITER ',' CSV HEADER;
COPY 4
```

```
loginord=# INSERT INTO stg_warehouses_raw (warehouse_code, warehouse_name, region, capacity) SELECT x.warehouse_code, x.warehouse_name, x.region, x.capacity::text FROM j
son_to_recordset(pg_read_file('/mnt/data/Documents/Doranco/Partiels/Février/warehouses_loginord.json')::json) AS x (warehouse_code TEXT, warehouse_name TEXT, region TEXT
, capacity int);
INSERT 0 3
```

4. Agréger les différents types de données (CSV, JSON, Excel) et unifier dans un schéma commun

Nous allons agréger les différentes données csv / xlsx / json et les unifier selon la granularité.

4.1 - Étape 1 : Définir la granularité cible

- Granularité recommandée : 1 ligne = 1 commande (table orders) + table de référence clients et warehouses

4.2 - Étape 2 : Normaliser les noms de colonnes

- Exemple : order_id, client_id, order_date, amount, warehouse_code
- Règle : snake_case, pas d'accents, pas d'espaces

4.3 - Étape 3 : Harmoniser formats et types

- Dates : convertir en YYYY-MM-DD (type DATE)
- Montants : convertir en numérique (type NUMERIC(10,2)), gérer les virgules et les points
- Entrepôt : référentiel stable (exemple : warehouse_code) pour éviter "Lille" / "LILLE" lille"

4.4 - Etape 4 : Gérer incohérence inter-sources

- Cas typiques : un client_id présent dans commandes mais absent du référentiel client ou un entrepôt "Roubaix" pas présent dans warehouses
- Deux stratégies possibles :
    - Créé une ligne "UNKOWN" et router les enregistrements
    - Ouvrir une liste d'anomalies à corriger côté métier (plus propre si c'est récurrent).

4.5 - Exemple de schéma final des tables :

- stg_orders_raw : table brute colonnes, texte, proche des fichiers), STG (staging) = zone de transit technique
- stg_clients_raw
- stg_warehouses_raw
- orders
- clients
- warehouses
- data_quality_issue

5. Valeurs manquantes : identification, analyse, correction (qualitative / quantitatives)

Nous allons maintenant identifier les valeurs manquantes, distinguer les variables qualitatives et quantitatives. Les valeurs manquantes ont été remplacées par la médiane (variables quantitatives) et par la modalité la plus fréquente (variables qualitatives).

5.1 - Étape 1 : Mesurer les manquants par variable avant correction

- Sur dataset (pandas) : compter isna() par colonne
- En SQL : compter WHERE col IS NULL par colonne

5.3 - Étape 2 : Classer les variables

- Quantitatives : amount, capacity …
- Qualitatives : warehouse, segment_client, ville

5.4 - Étape 3 : Choisir une stratégie justifiée

- **Quantitatives** :
  - Si il y a un montant manquant à cause d'une erreur de saisie : imputation par médiane ou moyenne si la distribution est stable
  - Si "0" a un sens métier : autoriser 0 mais distinguer "0 réel" / "manquant"
- **Qualitative** :
  - Imputation par modalité la plus fréquente si c'est cohérent
  - Sinon créer une modalité "UNKOWN"

Il faut également éviter de supprimer toutes les lignes, il faut forcément se justifier et ne pas faire de suppression massive.

6. Chargement des données nettoyées en garantissant l'intégrité

Nous allons maintenant comparer le nombre d'observations entre les datasets initiaux et les tables, vérifier le contenu et la lisibilité des éléments, vérifier les types.

6.1 - Étape 1 ; Charger d'abord en tables staging (brut)

- Exemple avec PostgreSQL : COPY stg_orders_raw FROM … CSV HEADER;
- L'objectif est de reproduire le fichier avant les transformations

6.2 - Étape 2 : Transformer vers les tables finales

- INSERT INTO orders ... SELECT … FROM stg_orders_raw … ;
- Cast types, normalisation, imputations

```
loginord=# CREATE TABLE orders (order_id INTEGER PRIMARY KEY, client_id INTEGER NOT NULL, order_date DATE NOT NULL, amount NUMERIC(10,2), warehouse VARCHAR(50) NOT NULL)
;
CREATE TABLE
loginord=# INSERT INTO orders(order_id, client_id, order_date,amount,warehouse) SELECT order_id::int,client_id::int,order_date::date,NULLIF(amount, '')::numeric(10,2),wa
rehouse FROM stg_orders_raw ON CONFLICT (order_id) DO NOTHING;
INSERT 0 5
```

6.3 - Étape 3 : Vérifier 100% du chargement

Requêtes types d'exemples :

Comptage des lignes :

```
loginord=# SELECT COUNT(*) FROM stg_orders_raw;
 count
-------
     5
(1 ligne)

loginord=# SELECT COUNT(*) FROM stg_clients_raw;
 count
-------
     4
(1 ligne)

loginord=# SELECT COUNT(*) FROM stg_warehouses_raw;
 count
-------
     3
(1 ligne)
```

Vérification du contenu :

```
loginord=# SELECT * FROM orders ORDER BY order_id LIMIT 10;
 order_id | client_id | order_date | amount | warehouse
----------+-----------+------------+--------+-----------
     1001 |       501 | 2025-01-05 |  89.90 | Lille
     1002 |       784 | 2025-01-05 |  39.00 | Roubaix
     1003 |       802 | 2025-01-06 |   0.00 | Tourcoing
     1004 |       501 | 2025-01-06 |        | Lille
     1005 |       912 | 2025-01-06 |  24.90 | Seclin
(5 lignes)
```

Vérification des types / valeurs :

```
loginord=# SELECT MIN(amount) AS min_amount, MAX(amount) AS max_amount, COUNT(*) FILTER (WHERE amount IS NULL) AS null_amounts FROM orders;
 min_amount | max_amount | null_amounts
------------+------------+--------------
       0.00 |      89.90 |            1
(1 ligne)
```

7. Types optimisés + longueur des champs (schéma brut / optimisé)

Nous allons maintenant optimiser en choisissant le type + nombre d'octets selon la longueur maximum des variables et selon le type du dataset initial.

A. Table brute

- stg_orders_raw(order_id TEXT, client_id, date TEXT, amount TEXT, warehouse TEXT)
- Avantages : charge facile, inconvénient : lent, conversions répétées, indexation moins efficace

B. Table optimisé (typée + contraintes)

- orders(order_id INT PRIMARY KEY, client_id INT NOT NULL, order_date DATE NOT NULL, amount NUMERIC(10,2) NOT NULL, warehouse_id SMALLINT NOT NULL)
- warehouse_id référence warehouses(id) et clients(id) …
- Longueurs : warehouse_name VARCHAR(50)

C. Indexation minimale pour la performance

- Index sur order_date, warehouse_id, client_id selon les requêtes métiers.

8. Comparaison des temps d'exécution : tables brutes vs optimisées

Nous allons comparer et mesurer le temps d'exécution entre tables non-optimisées et optimisées.

Étape 1 - Préparer 2 jeux de tables

- stg_orders_raw : brute
- orders : optimisée

Étape 2 - Choisir des requêtes représentatives

Exemples de requêtes "logistiques" :

1. Volume par jour :

```
loginord=# SELECT order_date, COUNT(*) FROM stg_orders_raw GROUP BY order_date;
 order_date | count
------------+-------
 2025-01-06 |     6
 2025-01-05 |     4
(2 lignes)

loginord=# SELECT order_date, COUNT(*) FROM orders GROUP BY order_date;
 order_date | count
------------+-------
 2025-01-05 |     2
 2025-01-06 |     3
(2 lignes)
```

2. Charge par entrepôt sur une période :

```
loginord=# SELECT warehouse, COUNT(*) FROM stg_orders_raw WHERE order_date BETWEEN '2025-01-01' AND '2025-01-31' GROUP BY warehouse;
 warehouse  | count
------------+-------
 Lille      |     4
 Roubaix    |     2
 Seclin     |     2
 Tourcoing  |     2
(4 lignes)

loginord=# SELECT warehouse, COUNT(*) AS nb_orders FROM orders WHERE order_date BETWEEN '2025-01-01' AND '2025-01-31' GROUP BY warehouse ORDER BY nb_orders DESC;
 warehouse  | nb_orders
------------+-----------
 Lille      |         2
 Roubaix    |         1
 Seclin     |         1
 Tourcoing  |         1
(4 lignes)
```

Étape 3 - Mesurer

- Dans psql : activer \timing puis exécuter les requêtes
- Ou alors utiliser : EXPLAIN ANALYZE SELECT …

Nous allons choisir la preuve avec \timing dans psql :

```
loginord=# \timing on
Chronométrage activé.
loginord=# SELECT warehouse, COUNT(*) AS nb_orders FROM stg_orders_raw WHERE order_date BETWEEN '2025-01-01' AND '2025-01-31' GROUP BY warehouse ORDER BY nb_orders DESC;
 warehouse  | nb_orders
------------+-----------
 Lille      |         4
 Roubaix    |         2
 Seclin     |         2
 Tourcoing  |         2
(4 lignes)

Temps : 1,091 ms
loginord=# SELECT warehouse, COUNT(*) AS nb_orders FROM orders WHERE order_date BETWEEN DATE '2025-01-01' AND DATE '2025-01-31' GROUP BY warehouse ORDER BY nb_orders DES
C;
 warehouse  | nb_orders
------------+-----------
 Lille      |         2
 Roubaix    |         1
 Seclin     |         1
 Tourcoing  |         1
(4 lignes)

Temps : 0,918 ms
loginord=#
```

La différence n’est pas significative vu que nous avons seulement 5 lignes, mais nous observons quand même 1,091 ms contre 0,918 ms entre la table brute et la table optimisée.

Étape 4 - Explication des écarts

- Table brute : conversions texte : date/nombre, group by sur texte, pas de clés / index adaptés
- Table optimisées : types natifs + index + jointures propres = moins de coût CPU / IO

9. Documentation des aspects RGPD, éthiques et les points de contrôle liés à la collecte et l'usage des données

Nous allons maintenant identifier la politique RGPD, le respect de la confidentialité et intégrer une dimension éthique ainsi que l'adaptation de communication selon les situations de handicap.

9.1 - Données personnelles

Tout identifiant client (client_id s'il est relié à une identité), nom / prénom, email, téléphone, adresse, IP, identifiants de livraison …

9.2 - Mesures concrètes

1. Minimisation : ne charger en base que les champs nécessaires aux objectifs (prévision de charge / allocation), éviter les champs inutiles
2. Pseudonymisation : séparer les tables d'identités et la table analytique. Utiliser des identifiants techniques (client_id) plutôt que des données direct
3. Contrôle d'accès : rôles SQL (lecture seule pour analystes, écritures limitée pour ETL), journaliser les accès si possible
4. Chiffrement : chiffrement disque / backup. TLS en transit si base sur serveur
5. Durée de conservation : définir une politique et une purge automatisée
6. Traçabilité qualité : table data_quality_issues pour suivre anomalies sans altérer l'historique

9.3 - Éthique, exemples de contrôles

1. Transparence d'usage : expliquer pourquoi les données sont utilisées (prévision de charge, optimisation)
2. Risque de décisions injustes : si l'allocation des ressources impacte les délais de livraison par zone, vérifier qu'il n'y a pas d'effets discriminants indirects.
3. "Human in the loop" : pour les anomalies détectées, prévoir une validation humaine sur les cas sensibles

9.4 - Accessibilité / handicap

Adapter les supports avec des compte-rendu clair, structuré et lisible, des canaux de communications selon les besoins, par exemple des documents compatibles, des contrastes adaptés …