

**Université des Sciences et de la Technologie Houari Boumediene**  
**USTHB**

## **Rapport de projet**

**Chaîne décisionnelle BI Northwind**  
**ETL Python – Data Warehouse SQLite – Dashboard Streamlit**

**Réalisé par : Zaidi Dina Meriem**

## 1. Introduction

Ce projet a pour objectif de mettre en œuvre une chaîne décisionnelle complète de Business Intelligence , depuis l'extraction de données issues de systèmes sources hétérogènes jusqu'à leur restitution via un tableau de bord interactif. Les données exploitées proviennent de la base de démonstration Northwind et sont disponibles sur deux environnements distincts : Microsoft Access et Microsoft SQL Server.

Ces deux sources présentent des différences en termes de structure (noms de colonnes, schémas), de volumétrie et de formats (dates, identifiants, chaînes). Une exploitation directe serait donc fragile et peu adaptée à l'analyse décisionnelle. Pour répondre à cette problématique, un processus ETL (Extract – Transform – Load) a été développé en Python afin de consolider et normaliser les données.

Les données transformées sont centralisées dans un Data Warehouse implémenté sous SQLite puis exploitées par un dashboard Streamlit.

### 1.1 Objectifs du projet

L'objectif principal est de concevoir et de mettre en œuvre une chaîne décisionnelle complète permettant l'analyse des commandes.

Plus précisément, le projet vise à :

- Intégrer des données provenant de Microsoft Access et de Microsoft SQL Server ;
- Nettoyer, harmoniser et consolider les données via un processus ETL automatisé ;
- Construire un Data Warehouse structuré selon une modélisation dimensionnelle (schéma en étoile) ;
- Proposer un dashboard interactif facilitant l'analyse des indicateurs clés de performance.

### 1.2 Périmètre du projet

Le périmètre fonctionnel est volontairement limité afin de garantir une mise en œuvre claire, testable et maîtrisée :

- Seules les tables directement liées aux commandes sont intégrées (Employees, Customers, Orders) ;
- L'analyse porte sur l'état des commandes (livrées / non livrées) et la performance des employés ;
- Les dimensions Produits et Fournisseurs ne sont pas intégrées dans cette version (limite assumée).

## 2. Architecture globale du système

L'architecture retenue suit une chaîne BI classique, structurée en couches afin de séparer clairement les responsabilités et de faciliter la maintenance :

Sources de données → ETL Python → Data Warehouse SQLite → Dashboard Streamlit.

Les sources Access et SQL Server alimentent l'ETL. L'ETL produit une base décisionnelle SQLite (tables de dimensions et table de faits), qui sert ensuite de source unique de vérité pour le dashboard.

### 2.2 Structure de l'arborescence

L'organisation des fichiers du projet suit une logique de couches (RAW → PROCESSED → FINAL) :

```
data/
  raw/           (sources brutes : exports CSV / fichiers Access)
  processed/    (fichiers intermédiaires : dimensions en CSV)
  final/         (livrables DWH : northwind_dw.sqlite, northwind_dw.xlsx, fact_orders.csv)
scripts/
  load_raw.py    (export RAW depuis SQL Server)
  etl.py         (ETL complet : extraction, transformation, chargement)
  dashboard.py   (interface de visualisation Streamlit + Plotly)
  sql.py         (chargement optionnel vers SQL Server DWH + script de restauration)
  Fact_Orders_Insert.sql (script SQL généré pour restaurer la table de faits)
```

## 3. Sources de données

Les données utilisées dans ce projet proviennent de la base Northwind. Les tables principales exploitées sont :

- Employees
- Customers
- Orders
- Order Details.

### 3.1 Exportation des tables SQL Server

Pour conserver une trace physique des données brutes issues de SQL Server, un script d'extraction exporte plusieurs tables vers des fichiers CSV dans la couche RAW. Le script se connecte à l'instance SQL Server, parcourt une liste de tables et génère un fichier par table avec un nom cohérent.

Exemple d'approche utilisée :

```

query = f"SELECT * FROM {table}"
df = pd.read_sql(query, conn)
df.to_csv(output_file, index=False, encoding="utf-8")

```

### 3.2 Volumétrie des données

Les deux systèmes sources n'ont pas la même volumétrie. Après consolidation, on obtient une vue globale plus complète.

Source	Employees	Customers	Orders
Microsoft Access	9	29	48
Microsoft SQL Server	9	91	830
Après consolidation	18	120	878

La consolidation permet de réunir les données des deux sources, tout en maintenant une traçabilité grâce à l'attribut source\_system, qui conserve l'origine (access / sqlserver).

### 3.3 Qualité des données sources

L'analyse des données sources met en évidence plusieurs problématiques classiques liées à l'hétérogénéité :

- Différences de nommage des colonnes entre Access et SQL Server ;
- Formats hétérogènes pour certains champs (dates, chaînes, identifiants) ;
- Présence de valeurs manquantes, notamment dans les dates d'expédition ;
- Risque de collision des identifiants métiers entre les deux systèmes (mêmes IDs possibles mais objets différents).

Ces constats justifient la mise en place d'un ETL robuste, avec normalisation des champs, clés de substitution (surrogate keys) et mapping basé sur une clé composite (source\_system, id\_métier).

## 4. Processus ETL (Extract – Transform – Load)

Le cœur du projet repose sur un ETL Python qui assure l'extraction depuis les sources, la transformation (nettoyage, harmonisation, consolidation) et le chargement dans le Data Warehouse SQLite. Le script ETL orchestre ces étapes et produit également des fichiers intermédiaires (CSV) et un livrable Excel.

### 4.1 Étape 1 – Connexion aux sources de données

Les connexions aux deux systèmes sources sont établies via ODBC à l'aide de pyodbc. Le script encapsule les paramètres de connexion dans des fonctions dédiées, ce qui facilite la maintenance et les modifications de configuration.

```

def conn_access():
    conn_str = (
        r"DRIVER={Microsoft Access Driver (*.mdb, *.accdb)};" +
        rf"DBQ={ACCESS_DB_PATH};"
    )
    return pyodbc.connect(conn_str)

```

## 4.2 Étape 2 – Extraction des données

L'extraction récupère les tables principales sous forme de DataFrames pandas. Côté Access, des tables optionnelles peuvent exister (Region, Territories, EmployeeTerritories) : elles sont chargées si disponibles, sinon remplacées par des DataFrames vides afin d'éviter les erreurs en aval.

Les tables principales extraites sont : Employees, Customers, Orders.

## 4.3 Étape 3 – Transformation des données

### 4.3.1 Harmonisation des colonnes

Les sources pouvant exposer des intitulés différents (ex. EmployeeID vs ID, FirstName vs First Name), le script applique une logique de renommage conditionnelle afin de converger vers un schéma commun. Cette approche permet de traiter des variantes sans casser le pipeline.

### 4.3.2 Nettoyage et normalisation des identifiants

Pour garantir un mapping fiable, les identifiants métiers sont normalisés (typage, suppression d'espaces, cohérence des champs). L'attribut source\_system est ajouté systématiquement afin de distinguer les enregistrements issus de systèmes différents.

### 4.3.3 Construction des dimensions (DimEmployee, DimCustomer)

Les dimensions sont obtenues par concaténation des données issues d'Access et de SQL Server. Chaque dimension reçoit une clé technique incrémentale (employee\_key, customer\_key), qui devient la référence interne du Data Warehouse.

Exemple pour les employés :

```

dim_emp = pd.concat([emp_a[cols], emp_s[cols]],
ignore_index=True).drop_duplicates()
dim_emp.insert(0, "employee_key", range(1, len(dim_emp) + 1))

```

Dans cette implémentation, un champ RegionDescription est conservé pour enrichir l'analyse (région ou valeur par défaut).

### 4.3.4 Construction de la dimension temporelle (DimDate)

La dimension date est générée indépendamment des sources, sur un intervalle large (1996–2030), afin de couvrir les dates de commande et d'expédition. Elle fournit des attributs utiles pour les analyses : année, mois, jour, nom du mois, jour de la semaine, week-end.

```
dim_date["date_key"] = dim_date["date"].dt.strftime("%Y%m%d").astype(int)
dim_date["day_of_week"] = dim_date["date"].dt.day_name()
dim_date["is_weekend"] = dim_date["day_of_week"].isin(["Saturday", "Sunday"])
```

#### 4.3.5 Construction de la table de faits (FactOrders)

La table de faits est construite à partir des commandes consolidées. Les dates sont converties en clés temporelles (order\_date\_key, ship\_date\_key), et les identifiants métiers (EmployeeID, CustomerID) sont mappés vers les clés techniques issues des dimensions.

Les mesures décisionnelles stockées sont :

- nb\_commandes\_livrees : vaut 1 si la date d'expédition est renseignée, sinon 0 ;
- nb\_commandes\_non\_livrees : vaut 1 si la date d'expédition est absente, sinon 0.

Sur la base des données consolidées, le Data Warehouse final contient 878 commandes, dont 848 livrées et 30 non livrées (taux de livraison ≈ 96.58 %).

#### 4.3.6 Mapping des clés techniques et intégrité référentielle

Pour éviter les collisions entre IDs métiers provenant de sources différentes, le mapping utilise une clé composite : (source\_system, id\_source). Ainsi, deux employés pouvant partager un même EmployeeID dans Access et SQL Server ne seront jamais confondus.

### 4.4 Étape 4 – Chargement dans le Data Warehouse (SQLite)

Les tables dimensionnelles et la table de faits sont chargées dans SQLite via la méthode pandas `to_sql()`, en mode remplacement pour garantir que chaque exécution génère un DWH cohérent.

```
conn = sqlite3.connect(DW_DB_PATH)
dim_emp.to_sql("dim_employee", conn, if_exists="replace", index=False)
fact_orders.to_sql("fact_orders", conn, if_exists="replace", index=False)
```

En complément, des livrables sont produits : un fichier Excel multi-feuilles (dimensions + faits) et un CSV de la table de faits, ce qui simplifie les vérifications et le partage des résultats.

### 4.5 Chargement final vers SQL Server et script de restauration

En option, le pipeline propose un chargement vers un SQL Server dédié (Northwind\_DWH).

Cette étape lit la table fact\_orders depuis SQLite, normalise les types (gestion des NULL, compatibilité DATETIME2) puis crée la table cible et insère les données en mode accéléré (fast\_executemany).

Enfin, un script SQL de restauration est généré automatiquement. Il contient la suite des instructions INSERT permettant de reconstruire la table FactOrders\_Final sans relancer l'ETL, améliorant la portabilité.

## 4.6 Contrôles et traçabilité

Les contrôles finaux consistent à vérifier : les volumes chargés, la répartition par source, et les indicateurs métier (livrées / non livrées). Ces contrôles garantissent la fiabilité de la couche décisionnelle. Des impressions console (et/ou logs) peuvent servir d'audit d'exécution.

# 5. Modélisation du Data Warehouse

Le Data Warehouse est modélisé selon un schéma en étoile (Star Schema). Cette modélisation sépare les mesures métier (table de faits) des descripteurs (dimensions), ce qui optimise les analyses multidimensionnelles et les requêtes d'agrégation.

## 5.1 Table de faits : fact\_orders

La table de faits fact\_orders constitue le cœur du modèle. Elle enregistre les événements liés aux commandes et contient les clés étrangères vers les dimensions ainsi que les indicateurs décisionnels.

Champs principaux :

- customer\_key : référence vers dim\_customer ;
- employee\_key : référence vers dim\_employee ;
- order\_date\_key : référence vers dim\_date pour la date de commande ;
- ship\_date\_key : référence vers dim\_date pour la date d'expédition ;
- nb\_commandes\_livrees / nb\_commandes\_non\_livrees : mesures décisionnelles.

## 5.2 Tables de dimensions

Les dimensions associées enrichissent la table de faits et facilitent l'analyse :

- dim\_employee : informations descriptives sur les employés (nom, fonction, localisation, région, source) ;
- dim\_customer : informations client (société, contact, localisation, téléphone, source) ;
- dim\_date : hiérarchies temporelles (année, mois, jour, week-end).

Volumes du DWH : dim\_employee = 18 lignes, dim\_customer = 120 lignes, dim\_date = 12784 lignes, fact\_orders = 878 lignes.

## 6. Dashboard décisionnel

### 6.1 Objectif du dashboard

Le dashboard a pour objectif de fournir une visualisation interactive des données décisionnelles issues du Data Warehouse. Il permet d'analyser l'état des commandes, d'évaluer la performance des employés et d'explorer les données selon une période donnée.

### 6.2 Connexion au Data Warehouse

Le dashboard se connecte à la base SQLite finale et exécute une requête de jointure entre la table de faits et les dimensions. Cela permet d'obtenir une vue enrichie directement exploitable dans l'interface (nom employé, client, région, date).

```
SELECT
    d.date AS order_date,
    e.employee_key,
    c.customer_key,
    TRIM(COALESCE(e.FirstName, '') || ' ' || COALESCE(e.LastName, '')) AS
employee_name,
    COALESCE(c.CompanyName, '(sans nom)') AS customer_name,
    COALESCE(e.RegionDescription, '(sans région)') AS region,
    f.nb_commandes_livrees,
    f.nb_commandes_non_livrees
FROM fact_orders f
LEFT JOIN dim_employee e ON f.employee_key = e.employee_key
LEFT JOIN dim_customer c ON f.customer_key = c.customer_key
LEFT JOIN dim_date d      ON f.order_date_key = d.date_key
```

### 6.3 Préparation des données pour l'analyse

Une fois les données chargées, le dashboard convertit order\_date au format datetime, élimine les dates invalides et s'assure que les KPI sont numériques. Des valeurs par défaut sont appliquées lorsque des informations sont manquantes (ex. employé inconnu, région non renseignée).

### 6.4 Mise en place des filtres interactifs

Le dashboard propose des filtres en barre latérale : sélection d'une période (date de commande) et sélection d'employés. Les filtres sont appliqués dynamiquement au DataFrame chargé afin de recalculer automatiquement les indicateurs et les graphiques.

### 6.5 Calcul des indicateurs affichés

Après application des filtres, les indicateurs clés sont recalculés : total des commandes, commandes livrées, commandes non livrées. Ces indicateurs reposent directement sur les champs de la table de faits, garantissant la cohérence avec l'ETL.

À titre indicatif (toutes périodes confondues), le DWH contient 878 commandes, 848 livrées et 30 non livrées.

## 6.6 Visualisation des données

Le dashboard combine plusieurs niveaux de restitution :

- un tableau détaillé (période × employé × client) ;
- une visualisation 3D permettant d'explorer la relation entre date, employé et client ;
- une courbe d'évolution mensuelle du volume de commandes.

La visualisation 3D est réalisée via `plotly.express.scatter_3d`. Les points sont réduits (marker size) afin de conserver une bonne lisibilité même en présence de nombreux clients. Le graphique mensuel (`plotly.express.line`) agrège les commandes par mois.

## 6.7 Interactivité et mise à jour dynamique

Toute modification des filtres entraîne une mise à jour instantanée des KPIs, du tableau et des graphiques. Cette approche offre une analyse flexible et adaptée aux besoins de l'utilisateur, sans nécessiter de recharge manuel.

# 7. Synthèse des choix et justifications techniques

## 7.1 Justification des bibliothèques (stack logicielle)

Les principales bibliothèques et technologies utilisées sont synthétisées ci-dessous :

Technologie	Rôle	Justification
Pandas	ETL & transformation	Traitement efficace sur colonnes (vectorisation), manipulation de DataFrames, export CSV/Excel.
PyODBC	Connectivité sources	Interopérabilité avec Access et SQL Server via drivers ODBC Microsoft.
SQLite	Data Warehouse	Base légère et portable (fichier unique), adaptée à un contexte académique.
Streamlit	Interface dashboard	Framework rapide pour construire une UI interactive avec filtres et KPIs.
Plotly	Visualisation	Graphiques interactifs (zoom, survol, rotation 3D) adaptés à

l'exploration.

Pathlib	Gestion des chemins	Compatibilité multi-systèmes et robustesse de l'arborescence projet.
---------	---------------------	--

## 7.2 Justifications architecturales (conception)

- Schéma en étoile : séparation faits/dimensions pour optimiser les agrégations et simplifier la compréhension du modèle.
- Clés de substitution : évitent les collisions entre identifiants métiers issus d'Access et de SQL Server.
- Dimension temporelle dédiée : évite de recalculer des attributs de date à la volée et simplifie les analyses chronologiques.

## 7.3 Justifications des méthodologies (logique ETL)

- Normalisation et harmonisation des colonnes : gestion de variantes de noms entre les systèmes sources.
- Mapping par clé composite (source\_system, id\_source) : garantit l'intégrité référentielle lors de la fusion.
- Approche batch : cohérence globale du DWH à chaque exécution, indicateurs recalculés sur une base certifiée.

## 7.4 Justifications de l'environnement

- Encapsulation des connexions : ouverture/fermeture rapide pour éviter les verrous prolongés sur Access et limiter les ressources côté SQL Server.
- Livrables multiples (SQLite + Excel + CSV) : facilite la validation, le partage et le débogage du pipeline.

## 7.5 Choix fonctionnels et limites assumées

- Absence de Slowly Changing Dimensions (SCD) : choix de simplification pour se concentrer sur l'état courant.
- Absence de certaines dimensions (produits, fournisseurs) : focalisation sur la commande et l'état de livraison.
- SQLite comme DWH : solution légère, portable et suffisante pour un contexte académique.

## 8. Conclusion

Ce projet met en œuvre une chaîne BI complète et opérationnelle : extraction et consolidation de données issues de sources hétérogènes, transformation et modélisation en schéma en étoile, chargement dans un Data Warehouse SQLite, et restitution via un dashboard interactif

Streamlit. L'ensemble assure une base décisionnelle fiable, cohérente et immédiatement exploitable pour l'analyse de l'état des commandes et le suivi de la performance des employés.