# Projet de Business Intelligence : Analyse des Performances Commerciales Northwind

## 1. Introduction

Ce projet a pour objectif de concevoir et de mettre en œuvre une plateforme complète de Business Intelligence (BI) à partir de la base de données transactionnelle classique **Northwind**. L'initiative vise à transformer des données opérationnelles brutes en tableaux de bord interactifs et en indicateurs clés de performance (KPI) actionnables pour la prise de décision . En structurant un processus ETL robuste et en modélisant un Data Warehouse adapté, le système fournit une vue consolidée et historique des performances commerciales. Les principaux résultats incluent l'automatisation des rapports, la réduction du temps d'accès à l'information et la mise à disposition d'analyses avancées sur le chiffre d'affaires, la profitabilité et le comportement client. La valeur décisionnelle de ce système réside dans sa capacité à identifier les tendances, à optimiser les stratégies de vente et à piloter la croissance commerciale sur des bases factuelles .

## 2. Objectifs détaillés

**Objectifs techniques**

1. Concevoir et implémenter un processus ETL automatisé et documenté.
2. Modéliser et peupler un Data Warehouse en schéma en étoile.
3. Développer une application de reporting interactif avec Streamlit.
4. Assurer la cohérence et la qualité des données entre la source et l'entrepôt.

**Objectifs analytiques**

1. Calculer et suivre des KPI commerciaux essentiels (CA, marge, quantités).
2. Analyser les performances par segment temporel, produit, catégorie et client.
3. Identifier les produits et clients les plus/le moins rentables.
4. Fournir une visualisation claire et exploitable des tendances.

**Objectifs pédagogiques**

1. Maîtriser le cycle complet d'un projet BI, de la source à la visualisation.
2. Appliquer les concepts théoriques de modélisation DW et d'ETL.
3. Développer des compétences en programmation Python, SQL et en visualisation de données.
4. Rédiger une documentation technique et fonctionnelle complète.

## 3. Architecture BI détaillée

L'architecture du projet suit une approche en couches classique et modulaire, garantissant la maintenabilité et l'évolutivité.

* Couche Source de données : La base de données transactionnelle « Northwind » sous SQL Server.

* Couche ETL (Extract, Transform, Load) : Un pipeline codé en « Python » (scripts `extract.py`, `transform.py`, `load.py`). Il est responsable de l'extraction incrémentielle, de l'application des règles de nettoyage et de transformation, et du chargement dans le DW.

* Couche Data Warehouse : Une base de données SQL Server dédiée, modélisée selon un « schéma en étoile » centré sur la table de fait `FactSales`.

* Couche Reporting & Visualisation : Deux interfaces complémentaires :

  * Une application web interactive développée avec « Streamlit » pour une analyse exploratoire.

  * Un tableau de bord statique conçu avec « Power BI » pour des rapports standardisés.

## 4. Organisation du projet

Project_Northwind_BI/

├── data/ # Gestion des données

│ ├── raw/ # Données brutes extraites (sauvegardes JSON/CSV)

│ └── processed/ # Données transformées prêtes au chargement

├── scripts/ # Cœur du pipeline ETL

│ ├── db.py # Module de connexion à la base de données

│ ├── extract.py # Script d'extraction et journalisation

│ ├── transform.py # Script de nettoyage et règles métier

│ └── load.py # Script de chargement dans le DW

├── reports/ # Génération de rapports PDF/HTML

├── Stat_PowerBI/ # Fichier Power BI (.pbix) et ses ressources

├── photos/ # Captures d'écran pour la documentation

├── extract-logs.txt # Fichier journal des exécutions ETL (succès/erreurs)

└── README.md # Documentation principale du projet

**Explication :**

* `data/raw/` et `data/processed/` : Permettent de suivre l'évolution des données et de déboguer le pipeline ETL.

* `scripts/` : Contient les modules Python modulaires pour chaque étape de l'ETL, favorisant la réutilisation du code.

* `extract-logs.txt` : Centralise les traces d'exécution, crucial pour le suivi et la gestion des erreurs.

* `README.md` : Documente l'objectif, l'installation, l'utilisation et l'architecture du projet.

## 5. Sources de données

Description de Northwind : Base de données classique simulant une entreprise de commerce de produits alimentaires. Elle contient des tables relatives aux clients, commandes, produits, employés, etc…

Tables OLTP utilisées : `Orders`, `Order Details`, `Products`, `Categories`, `Customers`, `Employees`, `Shippers`.

Granularité des données : La granularité la plus fine est la ligne de commande (un produit dans une commande).

## 6. Processus ETL (plus détaillé)

Le processus ETL est orchestré par des scripts Python spécifiques, exécutés dans l'ordre.

1. `db.py` (Connexion) :

   * Rôle : Module centralisé gérant les connexions aux bases de données source (Northwind OLTP) et cible (Data Warehouse) via SQLAlchemy.

   * Gestion des erreurs : Capture des exceptions de connexion (serveur indisponible, identifiants invalides) et journalisation.

2. `extract.py` (Extraction) :

   * Rôle : Interroge les tables sources de Northwind via des requêtes SQL. Pour des performances optimales, l'extraction est **incrémentielle** basée sur la date de la dernière modification (`ModifiedDate`).

   * Logs : Chaque extraction est enregistrée dans `extract-logs.txt` avec un timestamp, le nombre de lignes extraites par table, et le statut (succès/échec). Ce suivi est essentiel pour le contrôle du projet .

3. `transform.py` (Transformation) :

* Rôle : Applique les règles de nettoyage et les règles métier aux données extraites.

* Actions typiques : Nettoyage des textes (espaces, majuscules), gestion des valeurs NULL (imputation par "N/A" ou une valeur par défaut), conversion des types de données, calcul de colonnes dérivées (ex: `LigneTotal` = `UnitPrice` * `Quantity` * (1-`Discount`)).

* Validation : Vérifications d'intégrité (pas de doublons de clé, valeurs dans des plages attendues).

4. `load.py` (Chargement) :

* Rôle : Charge les données transformées dans les tables dimension et de fait du Data Warehouse.

* Stratégie : Pour les dimensions, vérification de l'existence (`UPSERT`). Pour les faits, insertion directe ou par lots.

* Gestion des erreurs : Transactions SQL pour assurer l'intégrité. En cas d'erreur lors du chargement d'un lot, la transaction est annulée (rollback) pour éviter un état incohérent.

## 7. Modélisation du Data Warehouse

Choix du Star Schema : Adopté pour sa simplicité, ses performances en requêtes et sa facilité de compréhension pour les utilisateurs métier.

Description :

* Table de Fait : `FactSales`

  * Mesures : `SalesAmount` (CA), `Quantity`, `DiscountAmount`, `UnitCost`, `Profit` (calculée).

  * Clés étrangères : `ProductKey`, `CustomerKey`, `EmployeeKey`, `OrderDateKey`, `ShipperKey`.

* Tables de Dimension :

  * `DimProduct` (Produits, Catégories)

  * `DimCustomer` (Clients, Région, Pays)

  * `DimEmployee` (Employés)

  * `DimDate` (Calendrier enrichi : année, trimestre, mois, jour de semaine)

  * `DimShipper` (Transporteurs)

Justification : Ce modèle répond directement aux objectifs analytiques. La dimension `DimDate` enrichie permet des analyses temporelles puissantes (YTD, YoY). La séparation des dimensions et des faits optimise les requêtes et simplifie l'ajout future de nouvelles dimensions.

## 8. Reporting & Visualisation

* Application Streamlit :

  * Architecture : Application web Python monolothique. Elle se connecte directement au Data Warehouse pour interroger les données à la volée.

  * KPI affichés : Tous les KPIs listés ci-dessus, présentés sous forme de métriques haute visibilité, graphiques (barres) et tableaux interactifs.

  * Interactivité : Filtres dynamiques par période, catégorie, pays, etc., permettant une analyse en "slice and dice".

* Tableau de bord Power BI :

  * Rôle secondaire : Utilisé pour créer une vue statique et très structurée, idéale pour des rapports opérationnels standardisés (ex: rapport hebdomadaire pour l'équipe de direction).

  * Fichier : `Stat_PowerBI/Stat_BI.pbix`

  * Validation visuelle : Vérification de la cohérence des chiffres entre Streamlit et Power BI pour assurer la fiabilité.

## 9. Conclusion

* Bilan technique : Le projet a permis de maîtriser l'ensemble de la stack technique d'un projet BI moderne, de la conception de l'architecture au déploiement d'applications de visualisation interactives.

* Bilan analytique : La modélisation du Data Warehouse et la définition des KPI ont renforcé la compréhension de la transformation des besoins métier en structures de données et en indicateurs concrets.

* Apports pédagogiques : Expérience pratique complète sur un cycle de projet, alliant théorie et mise en œuvre, et rencontrant des problèmes réels (qualité des données, performance ETL).

* Compétences acquises : Conception de Data Warehouse, développement de pipelines ETL en Python, modélisation de données, développement d'applications de visualisation (Streamlit, Power BI), gestion de projet et documentation.