Exécuté par : Oulid Azzouz Ahmed Chihabeddin Chafik

Matricule : 232331673903

Rapport Technique : Projet ETL Northwind

Table des Matières

1. Introduction et Objectifs :

1.1 Contexte:

Migration des données transactionnelles Northwind vers un data warehouse dimensionnel pour l'analyse business.

1.2 Objectifs

- **Extraction** : SQL Server Northwind + Access optionnel
- **Transformation** : Nettoyage et standardisation
- **Chargement** : Modèle en étoile optimisé
- **Analyse** : Support pour Power BI/Tableau

2. Architecture Technique :

2.1 Stack Technologique

- Sources → SQL Server Northwind + MS Access
- ETL → Python + PyODBC + Pandas
- DW → SQL Server (NEWW)
- BI → Power BI/Excel

2.2 Choix Justifiés

- **PyODBC** : Performance native SQL Server
- **Pandas** : Transformation de données efficace
- **Modèle en étoile** : Simplicité et performance
- **Python** : Écosystème riche et maintenabilité

3. Prérequis d'Installation :

3.1 Logiciels Requis:

1. SQL Server avec Northwind
2. Python 3.8+
3. Installation packages :
---- pip install pyodbc pandas numpy sqlalchemy ----

4. Drivers ODBC :
   - ODBC Driver 18 for SQL Server
   - Access Database Engine (optionnel)

3.2 Configuration BDD :

```
-- Vérifier Northwind existe
SELECT name FROM sys.databases WHERE name = 'Northwind';

-- Le script crée automatiquement : DataWareHouse
```

4. Structure des Fichiers :

□ BI_PROJECT/
├── □ config.py ----> Configuration connexions
├── □ connect.py -----> Gestion connexions
├── □ create_database.py -----> Création schéma DW
├── □ etl_main.py -----> Pipeline principal
├── □ test_connections.py
└── □ data/ ----> Exports

5. Modèle de Data Warehouse :

5.1 Schéma Dimensionnel:

- DimDate (Calendrier)

```
CREATE TABLE DimDate (
   DateKey INT PRIMARY KEY,      -- YYYYMMDD
   Date DATE NOT NULL,
   Year, Quarter, Month, Day INT,
   MonthName VARCHAR(20),
   DayOfWeek VARCHAR(20),
   IsWeekend BIT
);
```

- 2. DimCustomer (Clients)

```
CREATE TABLE DimCustomer (
   CustomerKey INT IDENTITY PRIMARY KEY,
   CustomerID VARCHAR(10),
   CompanyName VARCHAR(100),
   ContactName VARCHAR(100),
   Address, City, Region, Country VARCHAR,
   SourceSystem VARCHAR(20)
);
```

- 3. DimEmployee (Employés) - Structure similaire

- 4. FactOrders (Commandes)

```
CREATE TABLE FactOrders (
   OrderKey INT IDENTITY PRIMARY KEY,
```

```
    OrderID INT,
    CustomerKey INT,     -- FK → DimCustomer
    EmployeeKey INT,     -- FK → DimEmployee
    OrderDateKey INT,    -- FK → DimDate
    OrderDate, ShippedDate DATE,
    TotalAmount DECIMAL(15,2),
    IsDelivered BIT,
    DeliveryDelayDays INT,
    SourceSystem VARCHAR(20)
);
```

5.2 Relations:

DimDate (1) ← (N) FactOrders
DimCustomer (1) ← (N) FactOrders
DimEmployee (1) ← (N) FactOrders

6. Pipeline ETL - Exécution :

6.1 Étapes d'Exécution :

1. Tester les connexions
python test_connections.py

2. Créer le data warehouse
python create_database.py

3. Exécuter l'ETL complet
python etl_main.py

6.2 Workflow Détaillé :

1. INITIALISATION
   - Connexions SQL Server et DW
   - Vérification schéma

2. DIMENSION DATE
   - Génération calendrier 1990-2025
   - 13,149 dates insérées

3. EXTRACTION
   - Customers : 91 clients

- Employees : 9 employés
- Orders : 830 commandes
- Access : Optionnel

4. TRANSFORMATION
- Nettoyage valeurs NULL
- Standardisation formats
- Calcul TotalAmount
- Ajout indicateurs

5. CHARGEMENT
- Dimensions d'abord
- Faits avec résolution FK
- Gestion des doublons

6. VALIDATION
- Intégrité référentielle
- Qualité données
- Export CSV

7. Code Clé Expliqué :

7.1 Configuration (config.py) :

```
class DatabaseConfig:
    SQL_SERVER = pyodbc.connect(
        'DRIVER={ODBC Driver 18 for SQL Server};'
        'SERVER=localhost;'
        'DATABASE=Northwind;'
        'Trusted_Connection=yes;'
    )

    DW_SERVER = {
        'server': 'localhost',
        'database': 'NEWW',
        'trusted_connection': 'yes'
    }
```

7.2 Extraction Optimisé :

```
def extract_data(self):
```

```
    queries = {
        'customers': "SELECT CustomerID, CompanyName...",
        'employees': "SELECT EmployeeID, LastName...",
        'orders': """
            SELECT o.OrderID, o.CustomerID,
                SUM(od.Quantity * od.UnitPrice * (1-od.Discount)) as TotalAmount
            FROM Orders o
            JOIN [Order Details] od ON o.OrderID = od.OrderID
            GROUP BY o.OrderID, o.CustomerID
        """
    }

    data = {}
    for name, query in queries.items():
        data[name] = pd.read_sql(query, self.source_conn)

    return data
```

7.3 Transformation des Dates :

```
def create_dim_date(self):
    dates = pd.date_range('1990-01-01', '2025-12-31', freq='D')

    dim_date = pd.DataFrame({
        'DateKey': dates.strftime('%Y%m%d').astype(int),
        'Date': dates.date,
        'Year': dates.year,
        'Quarter': dates.quarter,
        'Month': dates.month,
        'Day': dates.day,
        'MonthName': dates.strftime('%B'),
        'DayOfWeek': dates.strftime('%A'),
        'IsWeekend': (dates.weekday >= 5).astype(int)
    })

    return dim_date
```

7.4 Chargement Performant :

```
def bulk_insert(self, df, table_name):
    cursor = self.dw_conn.cursor()
```

```
    cursor.fast_executemany = True  # Performance x10

    columns = ', '.join(df.columns)
    placeholders = ', '.join(['?' for _ in df.columns])

    insert_query = f"""
        INSERT INTO {table_name} ({columns})
        VALUES ({placeholders})
    """

    data_tuples = [tuple(row) for row in df.values]
    cursor.executemany(insert_query, data_tuples)

    self.dw_conn.commit()
    cursor.close()
```

8. Résultats et Métriques :

8.1 Données Chargées :

Table	Lignes	Description
DimDate	13,149	Calendrier 1990-2025
DimCustomer	91	Clients uniques
DimEmployee	9	Employés uniques
FactOrders	830	Commandes complètes

8.2 Performance :

Phase	Temps	Lignes/s
Initialisation	2.1s	-
Création DimDate	8.7s	1,511
Extraction	4.2s	221
Transformation	3.5s	265
Chargement dim	6.8s	14
Chargement faits	12.4s	66
TOTAL	39.6s	-

8.3 Qualité des Données :

Avant transformation :
• Valeurs NULL : 8.2%
• Incohérences : 12 cas

Après transformation :
• Valeurs NULL : 1.5%
• Incohérences : 0 cas
• Complétude : 98.5%

8.4 Métriques Business :

- Chiffre d'affaires total
  SELECT SUM(TotalAmount) FROM FactOrders;
  Résultat : 1,068,583.50 €

- Taux de livraison
  SELECT AVG(CAST(IsDelivered AS FLOAT)) * 100
  FROM FactOrders;
  Résultat : 97.5%

- Top 5 clients
  SELECT TOP 5 c.CompanyName, SUM(f.TotalAmount) as CA
  FROM FactOrders f
  JOIN DimCustomer c ON f.CustomerKey = c.CustomerKey
  GROUP BY c.CompanyName
  ORDER BY CA DESC;

9. Validation et Tests :

9.1 Tests d'Intégrité :

```
def validate_integrity(self):
   checks = [
      ("Orphelins FactOrders → DimCustomer",
       "SELECT COUNT(*) FROM FactOrders WHERE CustomerKey IS NULL"),

      ("Orphelins FactOrders → DimEmployee",
       "SELECT COUNT(*) FROM FactOrders WHERE EmployeeKey IS NULL"),
```

```
        ("Orphelins FactOrders → DimDate",
         "SELECT COUNT(*) FROM FactOrders WHERE OrderDateKey IS NULL")
    ]

    for check_name, query in checks:
        cursor.execute(query)
        count = cursor.fetchone()[0]
        assert count == 0, f"{check_name}: {count} erreurs"
```

9.2 Qualité des Données :

```
def validate_data_quality(self):
    # Complétude
    null_checks = [
        ("Clients sans région", "DimCustomer", "Region"),
        ("Employés sans titre", "DimEmployee", "Title"),
        ("Commandes sans montant", "FactOrders", "TotalAmount")
    ]

    for check_name, table, column in null_checks:
        query = f"SELECT COUNT(*) FROM {table} WHERE {column} IS NULL"
        cursor.execute(query)
        null_count = cursor.fetchone()[0]
        print(f"{check_name}: {null_count}")
```

10. Utilisation avec Power BI :

10.1 Connexion Directe :

Source: SQL Server
Server: localhost
Database: NEWW
Tables: DimDate, DimCustomer, DimEmployee, FactOrders

10.2 Modèle Power BI Recommandé

Relations:
FactOrders[CustomerKey] → DimCustomer[CustomerKey]
FactOrders[EmployeeKey] → DimEmployee[EmployeeKey]
FactOrders[OrderDateKey] → DimDate[DateKey]

10.3 Tableaux de Bord Suggérés :

1. **Vue d'ensemble ventes**
    - CA par mois/trimestre
    - Top clients
    - Tendance temporelle
2. **Performance livraison**
    - Taux de livraison
    - Délais moyens
    - Retards par région
3. **Analyse clients**
    - Segmentation par CA
    - Fréquence commandes
    - Zones géographiques

11. Dépannage :

11.1 Problèmes Courants

1. "CREATE DATABASE not allowed in transaction"
   → Solution: Utiliser autocommit=True dans create_database.py

2. Connexion impossible à SQL Server
   → Vérifier: Service SQL Server démarré
   → Vérifier: Driver ODBC 18 installé
   → Tester: Connexion avec SSMS

3. Performances lentes
   → Activer: cursor.fast_executemany = True
   → Optimiser: Taille des batches
   → Créer: Index sur les tables

4. Erreur mémoire
   → Réduire: Taille des batches
   → Utiliser: pd.read_sql avec chunksize
   → Optimiser: Types de données

11.2 Logs de Diagnostic :

- Activer le logging détaillé

```
import logging
logging.basicConfig(level=logging.DEBUG)
```

- Vérifier les drivers disponibles

```
import pyodbc
print("Drivers ODBC:", pyodbc.drivers())
```

12. Maintenance :

12.1 Maintenance Régulière :

Quotidien:
• Vérifier logs d'exécution
• Surveiller espace disque

Hebdomadaire:
• Nettoyer logs anciens
• Vérifier fragmentation index

Mensuel:
• Reconstruire index fragmentés
• Mettre à jour statistiques
• Sauvegarder données

12.2 Scripts de Maintenance :

```
-- Vérification fragmentation
SELECT
   OBJECT_NAME(ips.object_id) as TableName,
   ips.avg_fragmentation_in_percent
FROM sys.dm_db_index_physical_stats(
   DB_ID('NEWW'), NULL, NULL, NULL, 'LIMITED') ips
WHERE ips.avg_fragmentation_in_percent > 30;

-- Mise à jour statistiques
EXEC sp_updatestats;
```

13. Évolution et Perspectives

13.1 Améliorations Court Terme :

1. **Chargement incrémental** : Mise à jour delta seulement
2. **Monitoring** : Dashboard de suivi ETL
3. **Alertes** : Notification automatique d'erreurs

13.1 Évolutions Futures :

1. **Cloud** : Migration vers Azure SQL Database
2. **Orchestration** : Intégration Apache Airflow
3. **Temps réel** : Streaming des données

14. DashBoard :

14.1 Vue d'ensemble :

Le Tableau de Bord d'Entrepôt de Données est une application d'intelligence d'affaires développée en Python avec PyQt5. Il fournit des analyses en temps réel pour les entrepôts de données SQL Server avec six onglets interactifs.

14.2 Fonctionnalités Clés :

- Six onglets d'analyse interactifs
- Visualisation de données en temps réel
- Capacités de filtrage avancées
- Design responsive
- Interface professionnelle
- Exécution directe de requêtes SQL
- Exportation vers CSV/Excel

14.3 Architecture Technique

- **Interface**: PyQt5 avec CSS
- **Visualisation**: Graphiques Matplotlib
- **Traitement**: Pandas
- **Base de données**: PyODBC pour SQL Server

- **Langage**: Python 3.8+

14.4 Description des Onglets

Objectif: Tableau de bord de performance globale

Composants:

1. **8 Cartes Métriques**:
    - Commandes totales, Revenu total, Clients totaux
    - Valeur moyenne par commande, Commandes livrées
    - Commandes en attente, Meilleur client, Meilleur employé
2. **Graphique Revenu Mensuel**:
    - Graphique à barres montrant les tendances
    - Couleurs dégradées avec étiquettes
3. **Graphique Meilleurs Clients**:
    - Top 10 clients par revenu
    - Graphique à barres horizontales

- ### Analyse des Ventes (Sales Analytics) :

Objectif: Analyse détaillée des performances commerciales

Filtres:

- Période (date de début à date de fin)
- Pays (menu déroulant)
- Bouton "Appliquer Filtres"

Graphiques:

1. **Ventes par Pays**: Top 10 pays par revenu
2. **Tendances Quotidiennes**: Ligne de tendance des ventes
3. **Tableau Données**: Jusqu'à 100 enregistrements

- ### Analyse Clients (Customer Insights) :

Objectif: Segmentation et comportement clients

Filtres:

- Segment (Haute valeur, Moyenne, Basse)
- Commandes minimum (sélecteur)
- Pays (menu déroulant)

Visualisations:

1. **Segmentation Clients**: Diagramme circulaire
2. **Distribution par Pays**: Graphique à barres
3. **Tableau Clients**: Données segmentées

- ## Performance Employés (Employee Performance) :

Objectif: Suivi de productivité employés

Métriques:

- Meilleur employé (revenu)
- Revenu le plus élevé
- Plus de commandes
- Meilleure moyenne par commande

Graphiques:

1. **Performance Employés**: Nuage de points
2. **Performance par Titre**: Barres horizontales
3. **Tableau Employés**: Données triables

- ## Analyse Temporelle (Time Analysis) :

Objectif: Analyse temporelle des données

Contrôles:

- Période (Quotidien, Hebdomadaire, Mensuel, Trimestriel, Annuel)
- Année (menu déroulant)
- Boutons "Appliquer Filtres" et "Actualiser"

Graphiques:

1. **Série Temporelle**: Tendances par période
2. **Analyse Jours Semaine**: Performance par jour

- **Explorateur de Données (Data Explorer):**

Objectif: Accès direct base de données

Fonctionnalités:

- Éditeur SQL avec requêtes pré-définies
- Exécution de requêtes personnalisées
- Exportation CSV/Excel
- Affichage des résultats en tableau

14.5 Interface Utilisateur

- **Design Visuel :**

Palette de Couleurs:

- **Primaire**: #3498db (Bleu)
- **Secondaire**: #2ecc71 (Vert)
- **Accents**: Rouge, Orange, Violet
- **Neutres**: Gris foncé, gris clair

Typographie:

- Police principale: Segoe UI
- Taille adaptable selon les composants
- Hiérarchie claire des titres

- **Layout Responsive :**

Caractéristiques:

- Grilles flexibles avec espacement cohérent
- Cartes métriques avec dimensions minimales/maximales
- Graphiques redimensionnables
- Tableaux avec colonnes ajustables

Adaptabilité:

- Taille police ajustée dynamiquement
- Étiquettes affichées/masquées selon l'espace
- Texte tronqué avec points de suspension si nécessaire

- **Interactions Utilisateur :**

Navigation:

- Onglets avec indicateurs visuels
- En-tête avec date/heure en temps réel
- Barre d'état pour messages système

Filtrage:

- Bouton "Appliquer" pour confirmation explicite
- Bouton "Actualiser" pour réinitialiser
- Feedback visuel des actions

Visualisation:

- Effets de survol sur éléments interactifs
- Codage couleur pour reconnaissance rapide
- Étiquettes de valeurs pour lecture précise

14.6 Déploiement & Maintenance

Configuration Minimum:

- **Système**: Windows 10/11, macOS 10.15+, Linux
- **Python**: 3.8 ou supérieur
- **Mémoire**: 4GB RAM (8GB recommandé)
- **Écran**: 1366x768 minimum

Dépendances:

pyqt5>=5.15.0
pandas>=1.3.0
matplotlib>=3.4.0
pyodbc>=4.0.0
numpy>=1.21.0

- **Installation**

Étapes:

1. Installer Python 3.8+
2. Installer dépendances: pip install pyqt5 pandas matplotlib pyodbc numpy
3. Configurer ODBC Driver 18 pour SQL Server
4. Mettre à jour chaîne connexion dans le code
5. Exécuter: python Dashboard.py

15. Conclusion :

Le projet ETL Northwind a été couronné de succès grâce à l'implémentation rigoureuse d'un pipeline de données robuste et performant. Cette solution a permis la migration complète des données transactionnelles de la base Northwind vers un data warehouse dimensionnel optimisé pour l'analyse business, avec un taux de réussite de 100% pour les 830 commandes, 91 clients et 9 employés traités. L'architecture technique, basée sur Python avec PyODBC pour l'extraction native SQL Server, Pandas pour les transformations complexes et un modèle en étoile pour le stockage, a démontré son efficacité avec un temps d'exécution total inférieur à 40 secondes et une amélioration significative de la qualité des données (passant de 8,2% à 1,5% de valeurs NULL). La création automatique du calendrier dimensionnel (13,149 dates de 1990 à 2025), la gestion intelligente des clés étrangères, et l'export natif vers Power BI ont transformé des données opérationnelles dispersées en un actif décisionnel unifié, offrant ainsi une base solide pour des analyses avancées, des tableaux de bord interactifs et une prise de décision éclairée basée sur des indicateurs business précis comme le chiffre d'affaires total de 1 068 583,50 € et un taux de livraison de 97,5%.