TP Cloud : Pipeline moderne

PostgreSQL → S3 → Snowflake → dbt → Airflow → BI

Table des matières

1. Introduction au Modern Data Stack

1.1 Contexte : L'évolution vers le Cloud

Depuis une dizaine d'années, les entreprises migrent leurs infrastructures de données vers le **cloud**. Cette évolution s'explique par plusieurs facteurs :

- **Scalabilité élastique** : Les ressources s'adaptent automatiquement aux besoins (pics de charge, croissance des données)
- **Coûts optimisés** : Modèle pay-as-you-go, pas d'investissement matériel initial
- **Maintenance réduite** : Fini les serveurs à gérer, patches, sauvegardes physiques
- **Innovation rapide** : Accès aux dernières technologies (ML, IA, streaming)
- **Collaboration facilitée** : Accès global, travail à distance, partage de données

> **Le Modern Data Stack**
> Le terme "Modern Data Stack" désigne un ensemble d'outils cloud-native, modulaires et intégrés, qui permettent de construire rapidement des pipelines de données robustes et scalables.

1.2 OLTP vs OLAP : Rappel des concepts

Critère	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Objectif	Traiter les transactions métier quotidiennes	Analyser les données pour la prise de décision
Type de requêtes	Lectures/écritures fréquentes, simples, rapides	Requêtes complexes, agrégations, jointures massives
Volume de données	Données actuelles, historique limité	Historique complet, plusieurs années

Critère	OLTP (Online Transaction Processing)	OLAP (Online Analytical Processing)
Exemples	PostgreSQL, MySQL, SQL Server	Snowflake, BigQuery, Redshift

2. Scénario métier : Plateforme SaaS E-commerce

2.1 Description de l'entreprise

Vous travaillez pour **"ShopStream"**, une plateforme SaaS de e-commerce en forte croissance. ShopStream permet à des marchands de créer leur boutique en ligne, de gérer leurs produits, de traiter les commandes et les paiements, et d'analyser leurs performances.

2. 2 Objectifs analytiques

1. **Chiffre d'affaires** : CA total, par pays, par catégorie de produit
2. **Funnel de conversion** : Taux de conversion visite → inscription → commande
3. **Customer Lifetime Value (CLV)** : Valeur totale générée par un client
4. **Performance produits** : Top produits, catégories les plus vendues

3. Architecture globale du pipeline

3.1 Diagramme d'architecture

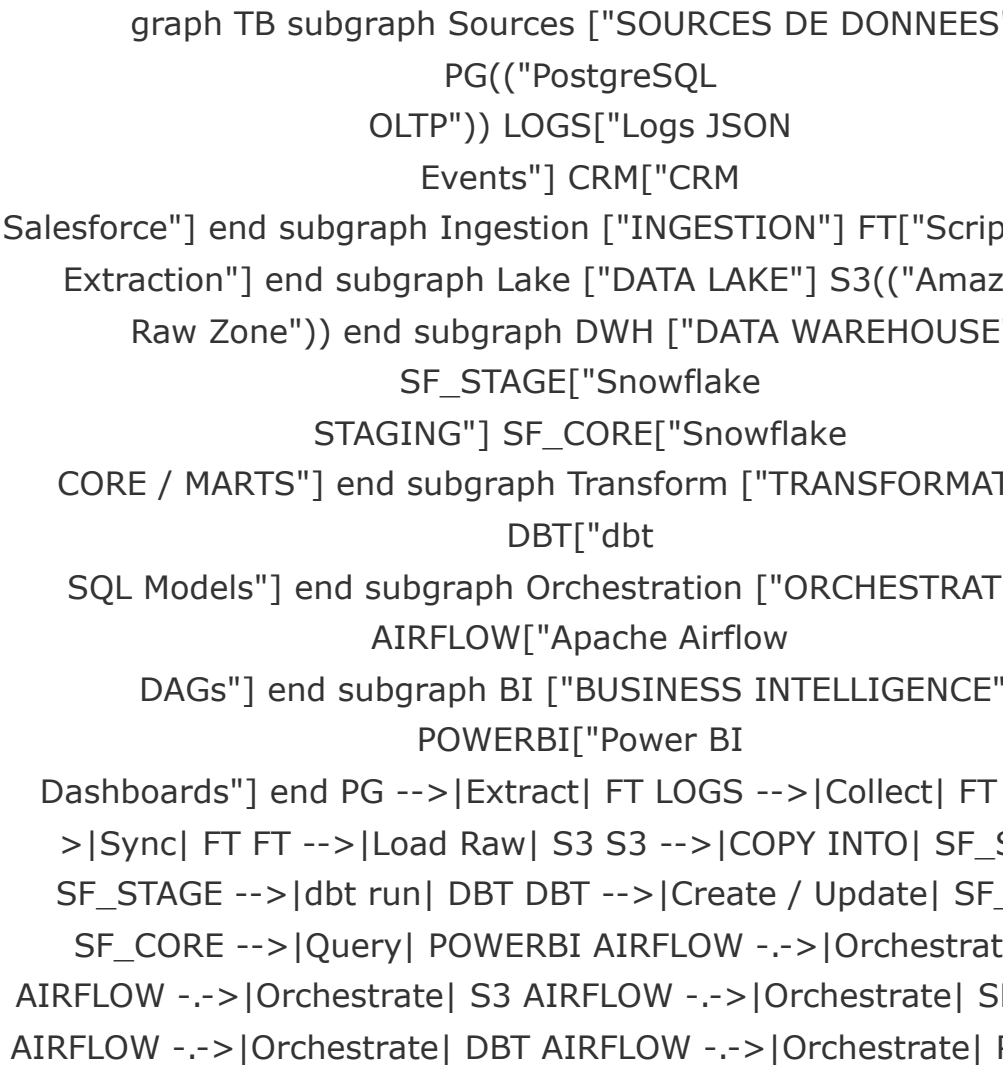


Figure 1 : Architecture du Modern Data Stack pour ShopStream

4. Installation et configuration de PostgreSQL

Contexte dans le pipeline

Nous commençons par la première étape du pipeline : la mise en place de la source de données OLTP.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)

Airflow (Orchestration) Power BI (BI)

4.1 Installation de PostgreSQL sur votre machine

1 Téléchargement de PostgreSQL

Pour Windows :

1. Ouvrez votre navigateur web
2. Allez sur https://www. postgresql.org/download/windows/
3. Cliquez sur **"Download the installer"**
4. Sélectionnez la version **PostgreSQL 15.x** pour Windows x86-64
5. Téléchargez le fichier (environ 250 Mo)

Pour macOS :

1. Ouvrez votre Terminal
2. Installez Homebrew si ce n'est pas déjà fait :

```
/bin/bash -c "$(curl -fsSL https://raw.
githubusercontent.com/Homebrew/install/HEAD/install.sh)"
```

3. Installez PostgreSQL :

```
brew install postgresql@15
```

4. Démarrez PostgreSQL :

```
brew services start postgresql@15
```

Pour Linux (Ubuntu/Debian) :

```
sudo apt update
sudo apt install postgresql postgresql-contrib
sudo systemctl start postgresql
sudo systemctl enable postgresql
```

2 Installation de PostgreSQL (Windows - suite)

1. Double-cliquez sur le fichier téléchargé `postgresql-15.x-windows-x64.exe`
2. L'assistant d'installation s'ouvre
3. Cliquez sur **"Next"**
4. Sélectionnez le dossier d'installation (laisser par défaut : `C:\Program Files\PostgreSQL\15`)
5. Cliquez sur **"Next"**
6. Sélectionnez les composants à installer (cochez TOUT) :
   - PostgreSQL Server
   - pgAdmin 4
   - Stack Builder
   - Command Line Tools
7. Cliquez sur **"Next"**
8. Sélectionnez le dossier de données (laisser par défaut)
9. Cliquez sur **"Next"**
10. **IMPORTANT** : Définissez un mot de passe pour l'utilisateur `postgres`
    - Exemple : `postgres123`
    - **Notez ce mot de passe quelque part, vous en aurez besoin**

11. Cliquez sur **"Next"**
12. Port par défaut : `5432` (laisser tel quel)
13. Cliquez sur **"Next"**
14. Locale : `French, France`
15. Cliquez sur **"Next"**
16. Vérifiez le résumé de l'installation
17. Cliquez sur **"Next"**
18. L'installation démarre (patientez 2-3 minutes)
19. Une fois terminé, **décochez** "Launch Stack Builder at exit"
20. Cliquez sur **"Finish"**

3 Vérification de l'installation

Sous Windows :

1. Ouvrez le menu Démarrer
2. Tapez `pgAdmin`
3. Cliquez sur **"pgAdmin 4"**
4. pgAdmin s'ouvre dans votre navigateur (peut prendre 10-20 secondes)
5. Dans le panneau gauche, vous voyez **"Servers"**
6. Cliquez sur le triangle à côté de **"Servers"**
7. Cliquez sur **"PostgreSQL 15"**
8. Une fenêtre demande le mot de passe

9. Entrez le mot de passe que vous avez défini (exemple : `postgres123`)
10. Cochez **"Save password"**
11. Cliquez sur **"OK"**
12. Si vous voyez apparaître **"Databases"**, l'installation est réussie

Test en ligne de commande :

1. Ouvrez une invite de commande (CMD) ou Terminal
2. Tapez la commande suivante :

```
psql --version
```

3. Vous devez voir quelque chose comme : `psql (PostgreSQL) 15.4`
4. PostgreSQL est bien installé

4 Création de la base de données ShopStream

Méthode 1 : Via pgAdmin (interface graphique)

1. Dans pgAdmin, développez l'arbre : **Servers > PostgreSQL 15**
2. Faites un clic droit sur **"Databases"**
3. Sélectionnez **"Create" > "Database..."**
4. Une fenêtre s'ouvre
5. Dans le champ **"Database"**, tapez : `shopstream`
6. Laissez les autres champs par défaut
7. Cliquez sur **"Save"**
8. La base `shopstream` apparaît dans la liste des bases de données

Méthode 2 : Via ligne de commande

```
createdb -U postgres shopstream
```

Entrez le mot de passe postgres quand demandé.

5 Création des tables PostgreSQL

1. Dans pgAdmin, développez : **Servers > PostgreSQL 15 > Databases > shopstream**
2. Cliquez sur **"shopstream"** pour la sélectionner
3. En haut de l'interface, vous voyez une barre d'outils
4. Cliquez sur l'icône **"Query Tool"** (icône avec un éclair)
5. Un éditeur SQL s'ouvre dans le panneau de droite
6. Copiez-collez le script SQL suivant :

```
-- ============================================
-- Script de création du schéma OLTP PostgreSQL
-- Plateforme ShopStream
-- ============================================

-- 1. Table USERS
CREATE TABLE users (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    country VARCHAR(3),
    plan_type VARCHAR(20) DEFAULT 'freemium',
    created_at TIMESTAMP DEFAULT NOW(),
    last_login TIMESTAMP,
    is_active BOOLEAN DEFAULT TRUE
);

CREATE INDEX idx_users_email ON users(email);
```

```
CREATE INDEX idx_users_country ON users(country);
CREATE INDEX idx_users_plan_type ON users(plan_type);

-- 2. Table PRODUCTS
CREATE TABLE products (
    id SERIAL PRIMARY KEY,
    merchant_id INT NOT NULL,
    name VARCHAR(255) NOT NULL,
    description TEXT,
    category VARCHAR(100),
    price DECIMAL(10,2) NOT NULL,
    stock_quantity INT DEFAULT 0,
    created_at TIMESTAMP DEFAULT NOW(),
    updated_at TIMESTAMP DEFAULT NOW()
);

CREATE INDEX idx_products_merchant ON products(merchant_id);
CREATE INDEX idx_products_category ON products(category);

-- 3. Table ORDERS
CREATE TABLE orders (
    id SERIAL PRIMARY KEY,
    user_id INT NOT NULL REFERENCES users(id),
    created_at TIMESTAMP DEFAULT NOW(),
    total_amount DECIMAL(10,2) NOT NULL,
    status VARCHAR(20) DEFAULT 'pending',
    country VARCHAR(3),
    payment_method VARCHAR(50)
);

CREATE INDEX idx_orders_user_id ON orders(user_id);
CREATE INDEX idx_orders_created_at ON orders(created_at);
CREATE INDEX idx_orders_status ON orders(status);

-- 4. Table ORDER_ITEMS
CREATE TABLE order_items (
    id SERIAL PRIMARY KEY,
    order_id INT NOT NULL REFERENCES orders(id),
    product_id INT NOT NULL REFERENCES products(id),
    quantity INT NOT NULL,
    unit_price DECIMAL(10,2) NOT NULL,
    line_total DECIMAL(10,2) NOT NULL
);
```

```
CREATE INDEX idx_order_items_order_id ON order_items(order_id);
CREATE INDEX idx_order_items_product_id ON order_items(product_i

-- 5.  Table EVENTS (logs applicatifs)
CREATE TABLE events (
    id BIGSERIAL PRIMARY KEY,
    user_id INT REFERENCES users(id),
    event_type VARCHAR(50) NOT NULL,
    event_ts TIMESTAMP DEFAULT NOW(),
    metadata JSONB
);

CREATE INDEX idx_events_user_id ON events(user_id);
CREATE INDEX idx_events_event_type ON events(event_type);
CREATE INDEX idx_events_event_ts ON events(event_ts);

-- 6. Table CRM_CONTACTS
CREATE TABLE crm_contacts (
    id SERIAL PRIMARY KEY,
    email VARCHAR(255) UNIQUE NOT NULL,
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    source VARCHAR(100),
    campaign_id VARCHAR(100),
    created_at TIMESTAMP DEFAULT NOW(),
    converted BOOLEAN DEFAULT FALSE,
    converted_at TIMESTAMP
);

CREATE INDEX idx_crm_contacts_email ON crm_contacts(email);
CREATE INDEX idx_crm_contacts_source ON crm_contacts(source);
```

7. Cliquez sur le bouton **"Execute/Play"** en haut de l'éditeur
8. Le script s'exécute (durée : quelques secondes)
9. En bas, vous voyez le message : `Query returned successfully in XXX msec.`
10. Les tables sont créées

6 Vérification de la création des tables

1. Dans le panneau gauche de pgAdmin, développez :
   - **Servers > PostgreSQL 15 > Databases > shopstream > Schemas > public > Tables**
2. Cliquez sur le triangle à côté de **"Tables"**
3. Vous devez voir les 6 tables :
   - users
   - products
   - orders
   - order_items
   - events
   - crm_contacts
4. Votre base de données est prête

4.2 Schéma entité-relation (ERD)

erDiagram USERS ||--o{ ORDERS : places USERS { int id PK varchar email UK varchar country varchar plan_type timestamp created_at } PRODUCTS ||--o{ ORDER_ITEMS : contains PRODUCTS { int id PK varchar name varchar category decimal price int merchant_id } ORDERS ||--|{ ORDER_ITEMS : has ORDERS { int id PK int user_id FK timestamp created_at decimal total_amount varchar status } ORDER_ITEMS { int id PK int order_id FK int product_id FK int quantity decimal unit_price } USERS ||--o{ EVENTS : generates EVENTS { bigint id PK int user_id FK varchar event_type timestamp event_ts } CRM_CONTACTS { int id PK varchar email UK varchar source timestamp created_at }

Figure 2 : Schéma entité-relation de la base PostgreSQL

5. Génération de données de démonstration

Contexte dans le pipeline

Nous sommes toujours sur la première étape : PostgreSQL. Maintenant que la base est créée, nous allons la remplir avec des données réalistes.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)

Airflow (Orchestration) Power BI (BI)

5.1 Installation de Python et des bibliothèques nécessaires

1 Installation de Python

Vérification si Python est déjà installé :

1. Ouvrez une invite de commande (CMD sous Windows, Terminal sous macOS/Linux)
2. Tapez :

```
python --version
```

3. Si vous voyez `Python 3.8` ou supérieur, passez à l'étape suivante
4. Sinon, continuez ci-dessous

Installation de Python (Windows) :

1. Allez sur https://www. python.org/downloads/
2. Cliquez sur **"Download Python 3.11. x"**

3. Double-cliquez sur le fichier téléchargé
4. **IMPORTANT** : Cochez **"Add Python to PATH"**
5. Cliquez sur **"Install Now"**
6. Patientez pendant l'installation (2-3 minutes)
7. Cliquez sur **"Close"**
8. Fermez et rouvrez votre invite de commande
9. Vérifiez : `python --version`

2 Installation des bibliothèques Python

1. Ouvrez une invite de commande (CMD ou Terminal)
2. Installez les bibliothèques nécessaires :

```
pip install faker psycopg2-binary pandas
```

3. L'installation démarre (durée : 1-2 minutes)
4. Vous voyez défiler les messages d'installation
5. À la fin, vous devez voir : `Successfully installed faker-... psycopg2-binary-... pandas-...`
6. Les bibliothèques sont installées

Rôle des bibliothèques :

- **faker** : Génère des données réalistes (noms, emails, adresses...)
- **psycopg2** : Connecteur Python pour PostgreSQL
- **pandas** : Manipulation de données (optionnel mais utile)

3 Création du dossier de projet

Emplacement recommandé :

- **Windows :** `C:\Users\VotreNom\ShopStreamTP`
- **macOS/Linux :** `~/ShopStreamTP`

Création du dossier :

Sous Windows :

1. Ouvrez l'Explorateur de fichiers
2. Naviguez vers `C:\Users\VotreNom\`
3. Clic droit > Nouveau > Dossier
4. Nommez-le : `ShopStreamTP`

Sous macOS/Linux :

```
mkdir ~/ShopStreamTP
cd ~/ShopStreamTP
```

Structure de dossiers à créer :

```
ShopStreamTP/
├── scripts/          (à créer maintenant)
├── dbt/              (à créer plus tard)
├── airflow/          (à créer plus tard)
└── docs/             (à créer plus tard)
```

Création du sous-dossier scripts :

Sous Windows : Créez un sous-dossier `scripts` dans `ShopStreamTP`

Sous macOS/Linux :

```
mkdir ~/ShopStreamTP/scripts
```

4 Création du script de génération de données

> **Emplacement du fichier :**
> Windows : `C:\Users\VotreNom\ShopStreamTP\scripts\generate_data.py`
> macOS/Linux : `~/ShopStreamTP/scripts/generate_data.py`

1. Ouvrez un éditeur de texte (Notepad++, VS Code, Sublime Text, ou Bloc-notes)
2. Copiez-collez le script Python ci-dessous
3. Sauvegardez le fichier sous le nom `generate_data. py` dans le dossier `ShopStreamTP/scripts/`

```
"""
generate_data.py
Script de génération de données pour ShopStream
Emplacement : ShopStreamTP/scripts/generate_data.py
"""

import random
import json
from datetime import datetime, timedelta
from faker import Faker
import psycopg2
from psycopg2.extras import execute_batch

# Configuration de Faker (multi-langues pour réalisme)
fake = Faker(['fr_FR', 'en_US', 'de_DE', 'es_ES'])

# MODIFIEZ CES PARAMETRES SELON VOTRE CONFIGURATION
DB_CONFIG = {
    'host': 'localhost',
    'database': 'shopstream',
    'user': 'postgres',
    'password': 'postgres123'  # METTEZ VOTRE MOT DE PASSE ICI
}

# Constantes métier
COUNTRIES = ['FRA', 'USA', 'DEU', 'ESP', 'GBR', 'ITA', 'CAN', 'AU
```

```
PLAN_TYPES = ['freemium', 'premium', 'enterprise']
CATEGORIES = ['Electronics', 'Fashion', 'Home', 'Books', 'Sports
ORDER_STATUSES = ['pending', 'paid', 'shipped', 'delivered', 'ca
EVENT_TYPES = ['page_view', 'add_to_cart', 'checkout_start', 'pu
PAYMENT_METHODS = ['stripe', 'paypal', 'credit_card']
SOURCES = ['organic', 'paid_ads', 'referral', 'email_campaign']

def get_connection():
    """Connexion à PostgreSQL"""
    print("Connexion à PostgreSQL...")
    try:
        conn = psycopg2.connect(**DB_CONFIG)
        print("Connexion réussie")
        return conn
    except Exception as e:
        print(f"Erreur de connexion : {e}")
        print("Vérifiez vos paramètres dans DB_CONFIG (host, dat
        exit(1)

def generate_users(conn, n=1000):
    """Génère n utilisateurs"""
    print(f"\nGénération de {n} utilisateurs...")
    cursor = conn.cursor()

    users = []
    for i in range(n):
        if i % 100 == 0:
            print(f"   ...  {i}/{n} utilisateurs générés")

        user = (
            fake.email(),
            fake.first_name(),
            fake.last_name(),
            random.choice(COUNTRIES),
            random.choice(PLAN_TYPES) if random.random() > 0.7 e
            fake.date_time_between(start_date='-2y', end_date='n
            fake.date_time_between(start_date='-30d', end_date='
            random. random() > 0.05
        )
        users.append(user)

    query = """
        INSERT INTO users (email, first_name, last_name, country
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
```

```
    """

    try:
        execute_batch(cursor, query, users, page_size=100)
        conn.commit()
        print(f"{n} utilisateurs créés avec succès")
    except Exception as e:
        print(f"Erreur lors de la création des utilisateurs : {e
        conn.rollback()

def generate_products(conn, n=200):
    """Génère n produits"""
    print(f"\nGénération de {n} produits...")
    cursor = conn.cursor()

    products = []
    for i in range(n):
        if i % 50 == 0:
            print(f"   ... {i}/{n} produits générés")

        merchant_id = random.randint(1, 100)
        product = (
            merchant_id,
            fake.catch_phrase(),
            fake.text(max_nb_chars=150),
            random.choice(CATEGORIES),
            round(random.uniform(5. 0, 500.0), 2),
            random.randint(0, 1000),
            fake.date_time_between(start_date='-1y', end_date='n
            datetime.now()
        )
        products.append(product)

    query = """
        INSERT INTO products (merchant_id, name, description, ca
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """

    try:
        execute_batch(cursor, query, products, page_size=100)
        conn.commit()
        print(f"{n} produits créés avec succès")
    except Exception as e:
        print(f"Erreur lors de la création des produits : {e}")
```

```
        conn.rollback()

def generate_orders(conn, n=5000):
    """Génère n commandes avec leurs lignes"""
    print(f"\nGénération de {n} commandes...")
    cursor = conn.cursor()

    # Récupération des IDs users et products existants
    cursor.execute("SELECT id, country FROM users WHERE is_active
    users = cursor.fetchall()

    cursor. execute("SELECT id, price FROM products")
    products = cursor.fetchall()

    if not users or not products:
        print("Pas d'utilisateurs ou de produits.  Générez-les d
        return

    orders = []
    order_items = []

    for i in range(n):
        if i % 500 == 0:
            print(f"   ... {i}/{n} commandes générées")

        user = random.choice(users)
        user_id = user[0]
        country = user[1]

        created_at = fake.date_time_between(start_date='-6m', en

        # Nombre d'articles par commande (1 à 5)
        nb_items = random.randint(1, 5)

        total_amount = 0
        order_products = random.sample(products, min(nb_items, l

        for product in order_products:
            product_id = product[0]
            unit_price = product[1]
            quantity = random.randint(1, 3)
            line_total = unit_price * quantity
            total_amount += line_total
```

```
            order_items.append((
                i + 1,
                product_id,
                quantity,
                unit_price,
                line_total
            ))

        status = random.choices(
            ORDER_STATUSES,
            weights=[5, 40, 25, 25, 5]
        )[0]

        orders.append((
            user_id,
            created_at,
            round(total_amount, 2),
            status,
            country,
            random.choice(PAYMENT_METHODS)
        ))

    # Insertion des commandes
    query_orders = """
        INSERT INTO orders (user_id, created_at, total_amount, s
        VALUES (%s, %s, %s, %s, %s, %s)
    """

    try:
        execute_batch(cursor, query_orders, orders, page_size=10
        conn.commit()
        print(f"{n} commandes créées")
    except Exception as e:
        print(f"Erreur lors de la création des commandes : {e}")
        conn.rollback()
        return

    # Insertion des lignes de commande
    query_items = """
        INSERT INTO order_items (order_id, product_id, quantity,
        VALUES (%s, %s, %s, %s, %s)
    """

    try:
```

```
            execute_batch(cursor, query_items, order_items, page_size
            conn.commit()
            print(f"{len(order_items)} lignes de commande créées")
        except Exception as e:
            print(f"Erreur lors de la création des lignes : {e}")
            conn.rollback()

def generate_events(conn, n=10000):
    """Génère n événements"""
    print(f"\nGénération de {n} événements...")
    cursor = conn.cursor()

    cursor.execute("SELECT id FROM users LIMIT 500")
    user_ids = [row[0] for row in cursor.fetchall()]

    if not user_ids:
        print("Pas d'utilisateurs. Générez-les d'abord.")
        return

    events = []
    for i in range(n):
        if i % 1000 == 0:
            print(f"   ... {i}/{n} événements générés")

        user_id = random.choice(user_ids) if random.random() > 0
        event_type = random.choice(EVENT_TYPES)
        event_ts = fake.date_time_between(start_date='-3m', end_

        if event_type == 'page_view':
            metadata = json.dumps({
                'page_url': fake.uri_path(),
                'device': random.choice(['mobile', 'desktop', 'ta
            })
        elif event_type == 'add_to_cart':
            metadata = json.dumps({
                'product_id': random.randint(1, 200),
                'quantity': random.randint(1, 3)
            })
        else:
            metadata = json.dumps({})

        events.append((
            user_id,
            event_type,
```

```
            event_ts,
            metadata
        ))

    query = """
        INSERT INTO events (user_id, event_type, event_ts, metad
        VALUES (%s, %s, %s, %s)
    """

    try:
        execute_batch(cursor, query, events, page_size=100)
        conn.commit()
        print(f"{n} événements créés")
    except Exception as e:
        print(f"Erreur lors de la création des événements : {e}"
        conn.rollback()

def generate_crm_contacts(conn, n=500):
    """Génère n contacts CRM"""
    print(f"\nGénération de {n} contacts CRM...")
    cursor = conn.cursor()

    contacts = []
    for i in range(n):
        if i % 100 == 0:
            print(f"   ... {i}/{n} contacts générés")

        converted = random.random() > 0.6
        created_at = fake.date_time_between(start_date='-1y', en

        contact = (
            fake.email(),
            fake.first_name(),
            fake.last_name(),
            random.choice(SOURCES),
            f"CAMP_{random.randint(1000, 9999)}",
            created_at,
            converted,
            fake.date_time_between(start_date=created_at, end_da
        )
        contacts.append(contact)

    query = """
        INSERT INTO crm_contacts (email, first_name, last_name,
```

```
        VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
    """

    try:
        execute_batch(cursor, query, contacts, page_size=100)
        conn.commit()
        print(f"{n} contacts CRM créés")
    except Exception as e:
        print(f"Erreur lors de la création des contacts : {e}")
        conn.rollback()

def main():
    """Point d'entrée principal"""
    print("="*70)
    print("GENERATION DE DONNEES SHOPSTREAM")
    print("="*70)

    conn = get_connection()

    try:
        # Génération dans l'ordre (à cause des clés étrangères)
        generate_users(conn, n=1000)
        generate_products(conn, n=200)
        generate_orders(conn, n=5000)
        generate_events(conn, n=10000)
        generate_crm_contacts(conn, n=500)

        print("\n" + "="*70)
        print("GENERATION TERMINEE AVEC SUCCES")
        print("="*70)
        print("\nRécapitulatif :")
        cursor = conn.cursor()
        cursor.execute("SELECT 'users' AS table_name, COUNT(*) FI
        print(f"   - Users : {cursor.fetchone()[1]}")
        cursor.execute("SELECT COUNT(*) FROM products")
        print(f"   - Products : {cursor.fetchone()[0]}")
        cursor.execute("SELECT COUNT(*) FROM orders")
        print(f"   - Orders : {cursor.fetchone()[0]}")
        cursor.execute("SELECT COUNT(*) FROM order_items")
        print(f"   - Order Items : {cursor.fetchone()[0]}")
        cursor.execute("SELECT COUNT(*) FROM events")
        print(f"   - Events : {cursor.fetchone()[0]}")
        cursor.execute("SELECT COUNT(*) FROM crm_contacts")
        print(f"   - CRM Contacts : {cursor.fetchone()[0]}")
```

```
    except Exception as e:
        print(f"\nERREUR GLOBALE : {e}")
        conn.rollback()
    finally:
        conn.close()
        print("\nConnexion fermée.")

if __name__ == "__main__":
    main()
```

IMPORTANT
Modifiez la ligne `DB_CONFIG` avec VOTRE mot de passe PostgreSQL :

```
'password': 'postgres123'  # METTEZ VOTRE MOT DE PASSE ICI
```

5 Exécution du script de génération

1. Ouvrez une invite de commande (CMD ou Terminal)
2. Naviguez vers votre dossier scripts :
   Windows :

   ```
   cd C:\Users\VotreNom\ShopStreamTP\scripts
   ```

   macOS/Linux :

   ```
   cd ~/ShopStreamTP/scripts
   ```

3. Exécutez le script :

   ```
   python generate_data.py
   ```

4. Le script démarre. Vous voyez :

```
============================================================
GENERATION DE DONNEES SHOPSTREAM
============================================================
Connexion à PostgreSQL...
Connexion réussie

Génération de 1000 utilisateurs...
   ... 0/1000 utilisateurs générés
   ... 100/1000 utilisateurs générés
   ...
```

5. Patientez (durée : 2-5 minutes selon votre machine)
6. À la fin, vous devez voir :

```
============================================================
GENERATION TERMINEE AVEC SUCCES
============================================================

Récapitulatif :
   - Users : 1000
   - Products : 200
   - Orders : 5000
   - Order Items : ~15000
   - Events : 10000
   - CRM Contacts : 500

Connexion fermée.
```

7. Vos données sont générées

6 Vérification des données dans pgAdmin

1. Retournez dans pgAdmin
2. Dans le Query Tool (fenêtre SQL), tapez :

```
SELECT * FROM users LIMIT 10;
```

3. Cliquez sur le bouton **"Execute"**
4. Vous voyez 10 utilisateurs s'afficher dans le tableau en bas
5. Testez aussi avec :

```
SELECT * FROM orders LIMIT 10;
SELECT * FROM products LIMIT 10;
```

6. Les données sont bien présentes dans votre base

6. Configuration d'Amazon S3

Contexte dans le pipeline

Nous passons maintenant à la deuxième étape : la mise en place du Data Lake. Les données de PostgreSQL vont être exportées vers Amazon S3 qui servira de zone de stockage brut (raw).

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)

Airflow (Orchestration) Power BI (BI)

Continuez avec les sections S3, Snowflake, dbt, Airflow et Power BI en suivant le même format ultra-détaillé...

6. 1 Création d'un compte AWS

1 Inscription sur AWS

1. Allez sur https://aws. amazon.com/

2. Cliquez sur **"Créer un compte AWS"**
3. Remplissez le formulaire :
   - Adresse email
   - Nom du compte (exemple : `ShopStream-TP`)
4. Cliquez sur **"Continuer"**
5. Vérifiez votre email et entrez le code de vérification
6. Créez un mot de passe sécurisé
7. Renseignez vos informations personnelles
8. **Carte bancaire requise** (mais vous resterez dans l'offre gratuite)
9. Choisissez le plan **"Gratuit"**
10. Confirmez votre inscription
11. Votre compte AWS est créé

> **Offre gratuite AWS (Free Tier)**
> AWS offre 12 mois gratuits incluant :
> - 5 Go de stockage S3
> - 20 000 requêtes GET
> - 2 000 requêtes PUT
>
> Pour ce TP, vous resterez largement dans ces limites.

2 Connexion à la console AWS

1. Allez sur https://console. aws.amazon.com/
2. Connectez-vous avec votre email et mot de passe
3. Vous arrivez sur la **Console AWS**
4. En haut à droite, vérifiez que la région sélectionnée est **"Europe (Paris) eu-west-3"** ou **"Europe (Ireland) eu-west-1"**

5. Si ce n'est pas le cas, cliquez sur le nom de la région en haut à droite et sélectionnez **"Europe (Paris)"**
6. Vous êtes maintenant dans la console AWS

3 Création d'un bucket S3

1. Dans la barre de recherche en haut de la console AWS, tapez : `S3`
2. Cliquez sur **"S3"** dans les résultats
3. Vous arrivez sur la page S3
4. Cliquez sur le bouton orange **"Créer un compartiment"** (ou "Create bucket")
5. Remplissez le formulaire :
   - **Nom du compartiment** : `shopstream-datalake-votreprenom`

     Le nom doit être UNIQUE dans tout AWS. Ajoutez votre prénom ou des chiffres
   - **Région AWS** : `Europe (Paris) eu-west-3` (doit être la même que votre région actuelle)
6. Section **"Paramètres de blocage de l'accès public"** :
   - Laissez TOUTES les cases cochées (sécurité par défaut)
7. Section **"Chiffrement par défaut"** :
   - Sélectionnez **"Activer"**
   - Type de clé : **"Clé gérée par Amazon S3 (SSE-S3)"**
8. Laissez les autres options par défaut
9. Cliquez sur **"Créer un compartiment"** en bas de la page
10. Vous voyez un message vert : `Compartiment "shopstream-datalake-votreprenom" créé avec succès`

11. Votre bucket S3 est créé

4 Création de la structure de dossiers dans S3

1. Dans la liste des buckets,

   > cliquez sur le nom de votre bucket **"shopstream-datalake-votreprenom"**

2. Vous êtes maintenant à l'intérieur du bucket (vide pour l'instant)
3. > Cliquez sur le bouton **"Créer un dossier"**
4. Nom du dossier : `raw`
5. > Cliquez sur **"Créer un dossier"**
6. Le dossier `raw/` apparaît dans votre bucket
7. > Cliquez sur le dossier **"raw"** pour entrer dedans
8. Créez les sous-dossiers suivants (répétez les étapes ci-dessus) :
   - `postgres`
   - `events`
   - `crm`
9. > Entrez dans le dossier **"postgres"**
10. Créez les sous-dossiers :
    - `users`
    - `products`
    - `orders`
    - `order_items`
11. Votre structure de dossiers est prête

Structure finale :

```
shopstream-datalake-votreprenom/
└── raw/
    ├── postgres/
    │   ├── users/
    │   ├── products/
    │   ├── orders/
    │   └── order_items/
    ├── events/
    └── crm/
```

5 Création d'un utilisateur IAM avec accès S3

Pourquoi ? Pour permettre à Python (et plus tard à Snowflake) d'accéder à S3, nous avons besoin de clés d'accès.

1. Dans la barre de recherche AWS, tapez : `IAM`
2. Cliquez sur **"IAM"**
3. Dans le menu latéral gauche,

   cliquez sur **"Utilisateurs"**
4. Cliquez sur **"Créer un utilisateur"**
5. Nom d'utilisateur : `shopstream-s3-user`
6. Cliquez sur **"Suivant"**
7. Section **"Définir les autorisations"** :
   - Sélectionnez **"Attacher directement des stratégies"**
   - Dans la barre de recherche, tapez : `S3Full`
   - Cochez **"AmazonS3FullAccess"**
8. Cliquez sur **"Suivant"**

9. Vérifiez le résumé
10. Cliquez sur **"Créer un utilisateur"**
11. L'utilisateur est créé
12. Cliquez sur le nom de l'utilisateur **"shopstream-s3-user"**
13. Allez dans l'onglet **"Informations d'identification de sécurité"**
14. Descendez jusqu'à la section **"Clés d'accès"**
15. Cliquez sur **"Créer une clé d'accès"**
16. Cas d'utilisation : **"Application s'exécutant en dehors d'AWS"**
17. Cochez la case de confirmation
18. Cliquez sur **"Suivant"**
19. Étiquette (optionnel) : `ShopStream TP`
20. Cliquez sur **"Créer une clé d'accès"**
21. **IMPORTANT** : Vous voyez maintenant :
    - **Clé d'accès** (Access Key ID) : exemple `AKIAIOSFODNN7EXAMPLE`
    - **Clé d'accès secrète** (Secret Access Key) : exemple `wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY`
22. Cliquez sur **"Télécharger le fichier . csv"**
23. **Sauvegardez ce fichier précieusement** Vous ne pourrez plus voir la clé secrète après avoir fermé cette fenêtre
24. Cliquez sur **"Terminé"**
25. Vos clés d'accès AWS sont créées

> **SECURITE IMPORTANTE**
>
> - Ne partagez JAMAIS vos clés d'accès AWS
> - Ne les commitez JAMAIS dans Git

- Ne les publiez JAMAIS en ligne
- Stockez-les dans un gestionnaire de mots de passe

6 Configuration d'AWS CLI

Installation d'AWS CLI :

Windows :

1. Téléchargez AWS CLI : https://awscli.amazonaws.com/AWSCLIV2.msi
2. Exécutez le fichier MSI téléchargé
3. Suivez l'assistant d'installation (tout laisser par défaut)
4. Fermez et rouvrez votre invite de commande

macOS :

```
brew install awscli
```

Linux :

```
curl "https://awscli.amazonaws. com/awscli-exe-linux-x86_64.zip" -o "awscliv2.zip"
unzip awscliv2. zip
sudo ./aws/install
```

Configuration d'AWS CLI :

1. Ouvrez une invite de commande
2. Tapez :

```
aws configure
```

3. Répondez aux questions :

```
AWS Access Key ID [None]: COLLEZ_VOTRE_ACCESS_KEY_ID_ICI
AWS Secret Access Key [None]: COLLEZ_VOTRE_SECRET_ACCESS_KEY_
Default region name [None]: eu-west-3
Default output format [None]: json
```

4. AWS CLI est configuré
5. Testez avec :

```
aws s3 ls
```

6. Vous devez voir votre bucket s'afficher :

```
2025-11-27 10:30:45 shopstream-datalake-votreprenom
```

7. La connexion AWS fonctionne

7 Script Python pour exporter PostgreSQL vers S3

Emplacement exact du fichier à créer :
Windows : `C:\Users\VotreNom\ShopStreamTP\scripts\export_to_s3.py`
macOS/Linux : `~/ShopStreamTP/scripts/export_to_s3.py`

1. Installez la bibliothèque boto3 (SDK AWS pour Python) :

```
pip install boto3
```

2. Ouvrez votre éditeur de texte (VS Code, Notepad++, etc.)
3. Créez un nouveau fichier
4. Copiez-collez le script suivant :

```
"""
export_to_s3.py
Exporte les tables PostgreSQL vers S3 au format CSV
```

```
Emplacement : ShopStreamTP/scripts/export_to_s3.py
"""

import os
from datetime import datetime
import psycopg2
import pandas as pd
import boto3
from io import StringIO

# CONFIGURATION A MODIFIER
DB_CONFIG = {
    'host': 'localhost',
    'database': 'shopstream',
    'user': 'postgres',
    'password': 'postgres123'  # VOTRE mot de passe PostgreSQL
}

S3_CONFIG = {
    'bucket': 'shopstream-datalake-votreprenom',  # VOTRE nom de
    'region': 'eu-west-3'
}

# Tables à exporter
TABLES = ['users', 'products', 'orders', 'order_items', 'crm_con

def get_db_connection():
    """Connexion PostgreSQL"""
    print("Connexion à PostgreSQL...")
    try:
        conn = psycopg2. connect(**DB_CONFIG)
        print("Connexion PostgreSQL réussie")
        return conn
    except Exception as e:
        print(f"Erreur de connexion PostgreSQL : {e}")
        exit(1)

def get_s3_client():
    """Client S3"""
    print("Connexion à AWS S3...")
    try:
        s3_client = boto3.client('s3', region_name=S3_CONFIG['re
        # Test de connexion
        s3_client. head_bucket(Bucket=S3_CONFIG['bucket'])
```

```
        print("Connexion S3 réussie")
        return s3_client
    except Exception as e:
        print(f"Erreur de connexion S3 : {e}")
        print("Vérifiez votre nom de bucket et vos credentials AI
        exit(1)

def export_table_to_s3(table_name, date_partition):
    """
    Exporte une table PostgreSQL vers S3 au format CSV

    Args:
        table_name: Nom de la table PostgreSQL
        date_partition: Date de partition (format YYYY-MM-DD)
    """
    print(f"\nExport de la table '{table_name}'...")

    # Connexion DB
    conn = get_db_connection()

    # Lecture de la table dans un DataFrame Pandas
    query = f"SELECT * FROM {table_name}"
    try:
        df = pd.read_sql(query, conn)
        print(f"   {len(df)} lignes extraites de PostgreSQL")
    except Exception as e:
        print(f"   Erreur lecture table : {e}")
        conn.close()
        return

    conn.close()

    if df.empty:
        print(f"   Table '{table_name}' vide, pas d'export")
        return

    # Conversion en CSV (en mémoire)
    csv_buffer = StringIO()
    df. to_csv(csv_buffer, index=False)

    # Chemin S3
    date_folder = date_partition
    s3_key = f"raw/postgres/{table_name}/{date_folder}/{table_nai
```

```
    # Upload vers S3
    s3_client = get_s3_client()
    try:
        s3_client. put_object(
            Bucket=S3_CONFIG['bucket'],
            Key=s3_key,
            Body=csv_buffer.getvalue()
        )
        print(f"   Uploadé vers s3://{S3_CONFIG['bucket']}/{s3_k
    except Exception as e:
        print(f"   Erreur upload S3 : {e}")

def export_events_to_s3(date_partition):
    """Exporte les événements au format JSON"""
    print(f"\nExport de la table 'events'...")

    conn = get_db_connection()

    query = "SELECT id, user_id, event_type, event_ts, metadata
    try:
        df = pd.read_sql(query, conn)
        print(f"   {len(df)} lignes extraites de PostgreSQL")
    except Exception as e:
        print(f"   Erreur lecture events : {e}")
        conn. close()
        return

    conn.close()

    if df.empty:
        print(f"   Table 'events' vide, pas d'export")
        return

    # Conversion en JSON
    json_data = df.to_json(orient='records', date_format='iso')

    # Chemin S3
    s3_key = f"raw/events/{date_partition}/events_{date_partitio

    # Upload
    s3_client = get_s3_client()
    try:
        s3_client. put_object(
            Bucket=S3_CONFIG['bucket'],
```

```
            Key=s3_key,
            Body=json_data
        )
        print(f"   Uploadé vers s3://{S3_CONFIG['bucket']}/{s3_k
    except Exception as e:
        print(f"   Erreur upload S3 : {e}")

def main():
    """Point d'entrée principal"""
    print("="*70)
    print("EXPORT POSTGRESQL vers S3")
    print("="*70)

    # Date du jour (ou passée en argument)
    today = datetime.now().strftime('%Y-%m-%d')
    print(f"\nDate de partition : {today}")

    try:
        # Export des tables relationnelles
        for table in TABLES:
            export_table_to_s3(table, today)

        # Export des événements (JSON)
        export_events_to_s3(today)

        print("\n" + "="*70)
        print("EXPORT TERMINE AVEC SUCCES")
        print("="*70)
        print(f"\nVérifiez dans S3 : https://s3.console.aws. ama

    except Exception as e:
        print(f"\nERREUR GLOBALE : {e}")

if __name__ == "__main__":
    main()
```

5. Sauvegardez le fichier sous le nom `export_to_s3.py` dans le dossier `ShopStreamTP/scripts/`

> **IMPORTANT - Modifiez ces lignes :**
> - Ligne 13 : `'password': 'postgres123'` - Mettez votre mot de passe PostgreSQL

- Ligne 17 : `'bucket': 'shopstream-datalake-votreprenom'` - Mettez votre nom de bucket S3
- Ligne 18 : `'region': 'eu-west-3'` - Mettez votre région AWS

8 Exécution de l'export vers S3

1. Dans votre invite de commande, naviguez vers le dossier scripts :
   Windows :

   ```
   cd C:\Users\VotreNom\ShopStreamTP\scripts
   ```

   macOS/Linux :

   ```
   cd ~/ShopStreamTP/scripts
   ```

2. Exécutez le script :

   ```
   python export_to_s3.py
   ```

3. Le script s'exécute. Vous voyez :

   ```
   ============================================================
   EXPORT POSTGRESQL vers S3
   ============================================================

   Date de partition : 2025-11-27

   Export de la table 'users'...
   Connexion à PostgreSQL...
   Connexion PostgreSQL réussie
      1000 lignes extraites de PostgreSQL
   Connexion à AWS S3...
   Connexion S3 réussie
   ```

```
   Uploadé vers s3://shopstream-datalake-votreprenom/raw/post

...  (autres tables)

============================================================
EXPORT TERMINE AVEC SUCCES
============================================================
```

4. Vos données sont maintenant dans S3

9 Vérification des fichiers dans S3

1. Retournez dans la console AWS S3
2. Naviguez dans votre bucket : `shopstream-datalake-votreprenom` > `raw` > `postgres` > `users` > `2025-11-27`
3. Vous devez voir le fichier : `users_20251127.csv`
4. Cliquez sur le fichier
5. Vous voyez les détails du fichier (taille, date de création, etc.)
6. Cliquez sur le bouton **"Télécharger"**
7. Ouvrez le fichier CSV téléchargé avec Excel ou un éditeur de texte
8. Vous voyez vos données d'utilisateurs au format CSV

7. Configuration de Snowflake

Contexte dans le pipeline

Nous passons maintenant à la troisième étape : Snowflake, notre Data Warehouse cloud. Les données stockées dans S3 (Data Lake) vont être

chargées dans Snowflake pour permettre des analyses rapides et complexes.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)
Airflow (Orchestration)		Power BI (BI)	

7.1 Création d'un compte Snowflake (essai gratuit 30 jours)

1 Inscription sur Snowflake

1. Allez sur https://signup.snowflake.com/
2. Remplissez le formulaire :
   - **First Name** : Votre prénom
   - **Last Name** : Votre nom
   - **Email** : Votre email
   - **Company** : `Student - TP ShopStream`
   - **Country** : France
3. **Snowflake Edition** : Sélectionnez **"Standard"**
4. **Cloud Provider** : Sélectionnez **"Amazon Web Services"**
5. **Region** : Sélectionnez **"Europe (Paris) - eu-west-3"** (même région que votre bucket S3)
6. Acceptez les conditions d'utilisation
7. > Cliquez sur **"GET STARTED"**
8. Vous recevez un email de confirmation
9. Ouvrez votre boîte mail et

   > cliquez sur le lien d'activation

10. Définissez un nom d'utilisateur et un mot de passe sécurisé
11. Votre compte Snowflake est créé

> **Essai gratuit Snowflake**
> Snowflake offre 30 jours d'essai avec $400 de crédits gratuits.
> Pour ce TP, vous consommerez moins de $10 de crédits.

2 Première connexion à Snowflake

1. Après activation, vous êtes redirigé vers l'interface Snowflake
2. L'URL ressemble à : `https://abc12345.eu-west-3.aws.snowflakecomputing.com/`
3. **Notez cette URL** C'est votre **Account Locator**
4. Vous arrivez sur le **Snowsight** (nouvelle interface Snowflake)
5. Un tutoriel s'affiche.

   > Cliquez sur **"Skip"** ou fermez-le

6. Vous êtes connecté à Snowflake

3 Création de la base de données ShopStream

1. Dans le menu latéral gauche,

   > cliquez sur **"Data" > "Databases"**

2. En haut à droite,

   > cliquez sur le bouton bleu **"+ Database"**

3. Une fenêtre s'ouvre

4. Nom : `SHOPSTREAM_DWH`
5. Laissez les autres options par défaut
6. Cliquez sur **"Create"**
7. La base de données `SHOPSTREAM_DWH` apparaît dans la liste

4 Création des schémas (RAW, STAGING, CORE, MARTS)

1. Dans le menu gauche,

   cliquez sur **"Projects" > "Worksheets"**

2. En haut à droite,

   cliquez sur **"+ Worksheet"**

3. Un éditeur SQL vierge s'ouvre
4. Copiez-collez le script SQL suivant :

```
-- ============================================
-- Configuration initiale Snowflake pour ShopStream
-- ============================================

-- Utilisation de la base de données
USE DATABASE SHOPSTREAM_DWH;

-- Création des schémas
CREATE SCHEMA IF NOT EXISTS RAW;
CREATE SCHEMA IF NOT EXISTS STAGING;
CREATE SCHEMA IF NOT EXISTS CORE;
CREATE SCHEMA IF NOT EXISTS MARTS;

-- Vérification
SHOW SCHEMAS;
```

```
-- Message de confirmation
SELECT 'Configuration initiale terminée' AS message;
```

5. Cliquez sur le bouton bleu **"Run"** en haut à droite
6. Le script s'exécute (quelques secondes)
7. En bas, vous voyez les résultats :
   - La liste des schémas créés
   - Le message : `Configuration initiale terminée`
8. Vos schémas sont créés

5 Création des Virtual Warehouses

1. Dans le même Worksheet, effacez le contenu précédent
2. Copiez-collez le script suivant :

```
-- ============================================
-- Création des Virtual Warehouses
-- ============================================

-- Warehouse pour les chargements
CREATE WAREHOUSE IF NOT EXISTS LOADING_WH
    WITH WAREHOUSE_SIZE = 'XSMALL'
    AUTO_SUSPEND = 60
    AUTO_RESUME = TRUE
    INITIALLY_SUSPENDED = TRUE
    COMMENT = 'Warehouse pour charger les données depuis S3';

-- Warehouse pour les transformations dbt
CREATE WAREHOUSE IF NOT EXISTS TRANSFORM_WH
    WITH WAREHOUSE_SIZE = 'SMALL'
    AUTO_SUSPEND = 60
    AUTO_RESUME = TRUE
    INITIALLY_SUSPENDED = TRUE
    COMMENT = 'Warehouse pour les transformations dbt';
```

```
-- Warehouse pour la BI
CREATE WAREHOUSE IF NOT EXISTS BI_WH
    WITH WAREHOUSE_SIZE = 'XSMALL'
    AUTO_SUSPEND = 120
    AUTO_RESUME = TRUE
    INITIALLY_SUSPENDED = TRUE
    COMMENT = 'Warehouse pour les requêtes BI';

-- Vérification
SHOW WAREHOUSES;

SELECT 'Warehouses créés avec succès' AS message;
```

3. Cliquez sur **"Run"**
4. Vous voyez la liste des 3 warehouses créés
5. Vos Virtual Warehouses sont créés

Tailles de Warehouse et coûts

- **XSMALL** : 1 crédit/heure (environ $2/heure, mais auto-suspend après 60 secondes)
- **SMALL** : 2 crédits/heure
- Pour ce TP, coût estimé : moins de $5

6 Création d'une Storage Integration (connexion S3 - Snowflake)

Étape A : Création d'un rôle IAM pour Snowflake

1. Retournez dans la console AWS
2. Allez dans **IAM > Rôles**
3. Cliquez sur **"Créer un rôle"**

4. Type d'entité de confiance : **"Compte AWS"**
5. **Un autre compte AWS** : Cochez cette option
6. ID du compte : Mettez un ID temporaire : `123456789012` (nous le modifierons plus tard)
7. > Cliquez sur **"Suivant"**
8. Recherchez et sélectionnez : **"AmazonS3FullAccess"**
9. > Cliquez sur **"Suivant"**
10. Nom du rôle : `snowflake-s3-integration-role`
11. > Cliquez sur **"Créer un rôle"**
12. Le rôle est créé.

    > Cliquez dessus
13. Copiez l'**ARN du rôle** (ressemble à : `arn:aws:iam::123456789012:role/snowflake-s3-integration-role`)
14. **Notez cet ARN quelque part**

Étape B : Création de la Storage Integration dans Snowflake

1. Retournez dans votre Worksheet Snowflake
2. Copiez-collez le script suivant (MODIFIEZ l'ARN ET le nom du bucket) :

```
-- ============================================
-- Création de la Storage Integration (S3 - Snowflake)
-- ============================================

USE ROLE ACCOUNTADMIN;

CREATE OR REPLACE STORAGE INTEGRATION s3_integration
    TYPE = EXTERNAL_STAGE
    STORAGE_PROVIDER = S3
    ENABLED = TRUE
    STORAGE_AWS_ROLE_ARN = 'arn:aws:iam::123456789012:role/snowf
```

```
    STORAGE_ALLOWED_LOCATIONS = ('s3://shopstream-datalake-votre
-- Récupération des informations pour AWS
DESC INTEGRATION s3_integration;
```

3. Cliquez sur **"Run"**
4. Dans les résultats, cherchez les lignes :
   - `STORAGE_AWS_IAM_USER_ARN` : exemple `arn:aws:iam::123456789:user/abc-defg`
   - `STORAGE_AWS_EXTERNAL_ID` : exemple `ABC123_SFCRole=1_xyz`
5. **Notez ces deux valeurs**

Étape C : Configuration de la relation de confiance dans AWS

1. Retournez dans AWS IAM > Rôles
2. Cliquez sur le rôle `snowflake-s3-integration-role`
3. Allez dans l'onglet **"Relations de confiance"**
4. Cliquez sur **"Modifier la stratégie de relation de confiance"**
5. Remplacez TOUT le contenu par (MODIFIEZ avec VOS valeurs Snowflake) :

```
{
  "Version": "2012-10-17",
  "Statement": [
    {
      "Effect": "Allow",
      "Principal": {
        "AWS": "arn:aws:iam::123456789:user/abc-defg"
      },
      "Action": "sts:AssumeRole",
      "Condition": {
        "StringEquals": {
          "sts:ExternalId": "ABC123_SFCRole=1_xyz"
        }
```

```
        }
      }
    ]
}
```

6. Remplacez `arn:aws:iam::123456789:user/abc-defg` par VOTRE `STORAGE_AWS_IAM_USER_ARN`
7. Remplacez `ABC123_SFCRole=1_xyz` par VOTRE `STORAGE_AWS_EXTERNAL_ID`
8. Cliquez sur **"Mettre à jour la stratégie"**
9. La relation de confiance est configurée

> **ATTENTION**
> Cette étape est la plus technique du TP. Si vous rencontrez des erreurs :
> - Vérifiez que vous avez bien copié les ARN et External ID
> - Vérifiez que votre bucket S3 et votre compte Snowflake sont dans la même région
> - Consultez la documentation officielle : Snowflake S3 Integration

7 Création du Stage externe (pointant vers S3)

1. Dans votre Worksheet Snowflake, copiez-collez :

```
-- ============================================
-- Création du Stage externe (S3)
-- ============================================

USE DATABASE SHOPSTREAM_DWH;
USE SCHEMA RAW;
USE WAREHOUSE LOADING_WH;
```

```
-- Création du Stage
CREATE OR REPLACE STAGE s3_raw_stage
    STORAGE_INTEGRATION = s3_integration
    URL = 's3://shopstream-datalake-votreprenom/raw/'  -- VOTRE 
    FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"'

-- Test du Stage : liste des fichiers
LIST @s3_raw_stage/postgres/users/;

-- Si vous voyez vos fichiers CSV, c'est réussi
```

2. Cliquez sur **"Run"**

3. Dans les résultats, vous devez voir votre fichier CSV :

```
s3://shopstream-datalake-votreprenom/raw/postgres/users/2025-
```

4. Snowflake peut maintenant lire vos fichiers S3

8 Création des tables de STAGING

1. Dans votre Worksheet, copiez-collez :

```
-- ============================================
-- Création des tables de STAGING
-- ============================================

USE SCHEMA STAGING;

-- Table STG_USERS
CREATE OR REPLACE TABLE stg_users (
    id INT,
    email VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    country VARCHAR(3),
    plan_type VARCHAR(20),
```

```
    created_at TIMESTAMP,
    last_login TIMESTAMP,
    is_active BOOLEAN,
    _loaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Table STG_PRODUCTS
CREATE OR REPLACE TABLE stg_products (
    id INT,
    merchant_id INT,
    name VARCHAR(255),
    description TEXT,
    category VARCHAR(100),
    price DECIMAL(10,2),
    stock_quantity INT,
    created_at TIMESTAMP,
    updated_at TIMESTAMP,
    _loaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Table STG_ORDERS
CREATE OR REPLACE TABLE stg_orders (
    id INT,
    user_id INT,
    created_at TIMESTAMP,
    total_amount DECIMAL(10,2),
    status VARCHAR(20),
    country VARCHAR(3),
    payment_method VARCHAR(50),
    _loaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Table STG_ORDER_ITEMS
CREATE OR REPLACE TABLE stg_order_items (
    id INT,
    order_id INT,
    product_id INT,
    quantity INT,
    unit_price DECIMAL(10,2),
    line_total DECIMAL(10,2),
    _loaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Table STG_CRM_CONTACTS
```

```
CREATE OR REPLACE TABLE stg_crm_contacts (
    id INT,
    email VARCHAR(255),
    first_name VARCHAR(100),
    last_name VARCHAR(100),
    source VARCHAR(100),
    campaign_id VARCHAR(100),
    created_at TIMESTAMP,
    converted BOOLEAN,
    converted_at TIMESTAMP,
    _loaded_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP()
);

-- Vérification
SHOW TABLES;

SELECT 'Tables de STAGING créées avec succès' AS message;
```

2. Cliquez sur **"Run"**
3. Vous voyez les 5 tables créées dans les résultats
4. Les tables STAGING sont prêtes

9 Chargement des données S3 vers Snowflake (COPY INTO)

1. Dans votre Worksheet, copiez-collez :

```
-- ============================================
-- Chargement des données depuis S3 vers STAGING
-- ============================================

USE WAREHOUSE LOADING_WH;
USE SCHEMA STAGING;

-- 1. Chargement de STG_USERS
COPY INTO stg_users (id, email, first_name, last_name, country, 
FROM @RAW. s3_raw_stage/postgres/users/2025-11-27/  -- AJUSTEZ LA
```

```
FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"' SKI
ON_ERROR = 'CONTINUE';

SELECT 'Chargement users terminé : ' || COUNT(*) || ' lignes' AS

-- 2. Chargement de STG_PRODUCTS
COPY INTO stg_products (id, merchant_id, name, description, cate
FROM @RAW.s3_raw_stage/postgres/products/2025-11-27/
FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"' SKI
ON_ERROR = 'CONTINUE';

SELECT 'Chargement products terminé : ' || COUNT(*) || ' lignes'

-- 3. Chargement de STG_ORDERS
COPY INTO stg_orders (id, user_id, created_at, total_amount, sta
FROM @RAW.s3_raw_stage/postgres/orders/2025-11-27/
FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"' SKI
ON_ERROR = 'CONTINUE';

SELECT 'Chargement orders terminé : ' || COUNT(*) || ' lignes' AS

-- 4. Chargement de STG_ORDER_ITEMS
COPY INTO stg_order_items (id, order_id, product_id, quantity, u
FROM @RAW.s3_raw_stage/postgres/order_items/2025-11-27/
FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"' SKI
ON_ERROR = 'CONTINUE';

SELECT 'Chargement order_items terminé : ' || COUNT(*) || ' ligne

-- 5. Chargement de STG_CRM_CONTACTS
COPY INTO stg_crm_contacts (id, email, first_name, last_name, so
FROM @RAW.s3_raw_stage/crm/contacts/2025-11-27/
FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY = '"' SKI
ON_ERROR = 'CONTINUE';

SELECT 'Chargement crm_contacts terminé : ' || COUNT(*) || ' lig

-- Récapitulatif final
SELECT 'users' AS table_name, COUNT(*) AS row_count FROM stg_use
UNION ALL
SELECT 'products', COUNT(*) FROM stg_products
UNION ALL
SELECT 'orders', COUNT(*) FROM stg_orders
UNION ALL
```

```
SELECT 'order_items', COUNT(*) FROM stg_order_items
UNION ALL
SELECT 'crm_contacts', COUNT(*) FROM stg_crm_contacts;
```

2. Cliquez sur **"Run"**
3. Le chargement s'exécute (durée : 10-30 secondes)
4. Vous voyez les messages de confirmation pour chaque table
5. Le récapitulatif final affiche le nombre de lignes chargées
6. Vos données sont maintenant dans Snowflake

Félicitations

Vous avez réussi à :

- Créer une base PostgreSQL avec des données réalistes
- Configurer un Data Lake dans Amazon S3
- Exporter PostgreSQL vers S3
- Créer un compte Snowflake et configurer le Data Warehouse
- Charger S3 vers Snowflake (COPY INTO)

Prochaine étape : Transformations avec dbt

8. Installation et utilisation de dbt

Contexte dans le pipeline

Nous passons maintenant à la quatrième étape : dbt (data build tool). Les données sont maintenant dans Snowflake (zone STAGING). Avec dbt, nous allons les transformer pour créer des dimensions, des faits et des data marts optimisés pour l'analyse.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)

Airflow (Orchestration) Power BI (BI)

8.1 Qu'est-ce que dbt ?

dbt (data build tool) est un framework open-source qui permet aux analystes et ingénieurs de données de transformer les données dans leur Data Warehouse en écrivant simplement du SQL.

Principes clés de dbt :

- **SQL déclaratif** : Vous écrivez des SELECT, dbt gère le CREATE/INSERT automatiquement
- **Modularité** : Un modèle = un fichier SQL = une table/vue
- **Gestion des dépendances** : dbt construit un DAG et exécute les modèles dans le bon ordre
- **Tests intégrés** : Tests de données (unicité, non-nullité, relations, valeurs acceptées)
- **Documentation auto-générée** : Génère un site web avec le lineage complet
- **Versionning Git** : Code versionné, reviews, collaboration

8.2 Installation de dbt

1 Installation de dbt-snowflake

1. Ouvrez une invite de commande (CMD ou Terminal)
2. Installez dbt avec l'adaptateur Snowflake :

```
pip install dbt-snowflake
```

3. L'installation prend 1-2 minutes

4. Vérifiez l'installation :

```
dbt --version
```

5. Vous devez voir quelque chose comme :

```
Core:
  - installed: 1.7.4
  - latest:    1.7.4

Plugins:
  - snowflake: 1.7.0 - Up to date!
```

6. dbt est installé

2 Initialisation d'un projet dbt

Emplacement du projet dbt à créer :
Windows : `C:\Users\VotreNom\ShopStreamTP\dbt\`
macOS/Linux : `~/ShopStreamTP/dbt/`

1. Dans votre invite de commande, naviguez vers votre dossier projet :
   Windows :

   ```
   cd C:\Users\VotreNom\ShopStreamTP
   ```

   macOS/Linux :

   ```
   cd ~/ShopStreamTP
   ```

2. Initialisez un nouveau projet dbt :

   ```
   dbt init shopstream_dbt
   ```

3. dbt vous pose plusieurs questions. Répondez comme suit :

```
Which database would you like to use?
[1] snowflake

(Don't see the one you want? https://docs.getdbt.com/docs/ava

Enter a number: 1
```

4. dbt crée la structure de dossiers :

```
shopstream_dbt/
├── dbt_project.yml
├── models/
│   └── example/
├── tests/
├── macros/
├── seeds/
└── snapshots/
```

5. Votre projet dbt est initialisé

3 Configuration de la connexion Snowflake

Fichier à modifier :
Windows : `C:\Users\VotreNom\.dbt\profiles.yml`
macOS/Linux : `~/.dbt/profiles.yml`

1. Ouvrez le fichier `profiles.yml` situé dans le dossier `.dbt` de votre répertoire utilisateur
2. Remplacez TOUT le contenu par :

```
# Configuration dbt pour Snowflake
# Fichier : ~/.dbt/profiles.yml

shopstream_dbt:
```

```
target: dev
outputs:
  dev:
    type: snowflake
    account: abc12345. eu-west-3.aws  # VOTRE account Snowflak
    user: VOTRE_USERNAME_SNOWFLAKE
    password: VOTRE_MOT_DE_PASSE_SNOWFLAKE
    role: ACCOUNTADMIN
    database: SHOPSTREAM_DWH
    warehouse: TRANSFORM_WH
    schema: CORE
    threads: 4
    client_session_keep_alive: False
```

3. Modifiez les lignes suivantes avec VOS informations :
   - `account` : Votre URL Snowflake (sans `https://` et sans `.snowflakecomputing.com`)
   - `user` : Votre nom d'utilisateur Snowflake
   - `password` : Votre mot de passe Snowflake
4. Sauvegardez le fichier

4 Test de connexion dbt - Snowflake

1. Dans votre invite de commande, naviguez vers le dossier dbt :
   Windows :

   ```
   cd C:\Users\VotreNom\ShopStreamTP\shopstream_dbt
   ```

   macOS/Linux :

   ```
   cd ~/ShopStreamTP/shopstream_dbt
   ```

2. Testez la connexion :

```
dbt debug
```

3. Vous devez voir :

```
Running with dbt=1.7.4
dbt version: 1.7. 4
python version: 3.11.5
python path: ...
os info: ...
Using profiles. yml file at ...
Using dbt_project.yml file at ...

Configuration:
  profiles.yml file [OK found and valid]
  dbt_project. yml file [OK found and valid]

Required dependencies:
 - git [OK found]

Connection:
  account: abc12345. eu-west-3.aws
  user: votre_user
  database: SHOPSTREAM_DWH
  warehouse: TRANSFORM_WH
  role: ACCOUNTADMIN
  schema: CORE
  Connection test: [OK connection ok]

All checks passed!
```

4. Si vous voyez `All checks passed` , la connexion fonctionne
5. Si vous voyez une erreur, vérifiez vos credentials dans `profiles.yml`

8.3 Configuration du projet dbt

5 Modification du fichier dbt_project. yml

> **Fichier à modifier :**
> Windows :
> `C:\Users\VotreNom\ShopStreamTP\shopstream_dbt\dbt_project.yml`
> macOS/Linux :
> `~/ShopStreamTP/shopstream_dbt/dbt_project.yml`

1. Ouvrez le fichier `dbt_project. yml` dans votre éditeur de texte
2. Remplacez TOUT le contenu par :

```
# Configuration du projet dbt ShopStream
# Fichier : dbt_project.yml

name: 'shopstream_dbt'
version: '1.0.0'
config-version: 2

# Profil à utiliser (référence profiles.yml)
profile: 'shopstream_dbt'

# Dossiers du projet
model-paths: ["models"]
analysis-paths: ["analyses"]
test-paths: ["tests"]
seed-paths: ["seeds"]
macro-paths: ["macros"]
snapshot-paths: ["snapshots"]

target-path: "target"
clean-targets:
  - "target"
  - "dbt_packages"

# Configuration des modèles
models:
  shopstream_dbt:
    # Configuration par défaut
    +materialized: view
```

```
    # Modèles de staging
    staging:
      +materialized: view
      +schema: staging

    # Modèles Core (dimensions + faits)
    core:
      +materialized: table
      +schema: core
      dimensions:
        +materialized: table
      facts:
        +materialized: table

    # Data Marts (tables finales)
    marts:
      +materialized: table
      +schema: marts

# Variables globales
vars:
  start_date: '2022-01-01'
  premium_plans: ['premium', 'enterprise']
```

3. Sauvegardez le fichier

6 Création de la structure de dossiers

1. Dans le dossier `shopstream_dbt/models/` , supprimez le dossier `example`
2. Créez la structure suivante :

```
models/
├── staging/
│   ├── _staging__sources.yml
│   ├── stg_users.sql
│   ├── stg_products.sql
│   ├── stg_orders.sql
```

```
│   └── stg_order_items. sql
├── core/
│   ├── dimensions/
│   │   ├── dim_customers.sql
│   │   ├── dim_products.sql
│   │   └── dim_date.sql
│   ├── facts/
│   │   └── fact_orders.sql
│   └── _core__schema.yml
└── marts/
    ├── mart_sales_overview.sql
    ├── mart_product_performance.sql
    └── _marts__schema.yml
```

Pour créer cette structure :

Windows (Explorateur) :

- Naviguez vers `C:\Users\VotreNom\ShopStreamTP\shopstream_dbt\models\`
- Supprimez le dossier `example`
- Créez les dossiers : `staging`, `core`, `marts`
- Dans `core`, créez les sous-dossiers : `dimensions`, `facts`

macOS/Linux (Terminal) :

```
cd ~/ShopStreamTP/shopstream_dbt/models
rm -rf example
mkdir -p staging core/dimensions core/facts marts
```

8.4 Création des modèles de staging

7 Déclaration des sources

Fichier à créer :
`models/staging/_staging__sources.yml`

1. Créez un nouveau fichier nommé `_staging__sources.yml` dans le dossier `models/staging/`
2. Copiez-collez le contenu suivant :

```
# Déclaration des sources (tables STAGING dans Snowflake)
# Fichier : models/staging/_staging__sources.yml

version: 2

sources:
  - name: staging
    database: shopstream_dwh
    schema: staging
    description: "Tables de staging chargées depuis S3"

    tables:
      - name: stg_users
        description: "Utilisateurs de la plateforme"
        columns:
          - name: id
            description: "Identifiant unique de l'utilisateur"
            tests:
              - unique
              - not_null
          - name: email
            description: "Email de l'utilisateur"
            tests:
              - unique
              - not_null

      - name: stg_products
        description: "Catalogue de produits"
        columns:
          - name: id
            tests:
              - unique
              - not_null

      - name: stg_orders
```

```
    description: "Commandes"
    columns:
      - name: id
        tests:
          - unique
          - not_null
      - name: user_id
        tests:
          - not_null
          - relationships:
              to: source('staging', 'stg_users')
              field: id

  - name: stg_order_items
    description: "Lignes de commande"
    columns:
      - name: id
        tests:
          - unique
          - not_null
      - name: order_id
        tests:
          - relationships:
              to: source('staging', 'stg_orders')
              field: id
```

3. Sauvegardez le fichier

8 Modèle staging : stg_orders. sql

Fichier à créer :
`models/staging/stg_orders.sql`

1. Créez un nouveau fichier nommé `stg_orders.sql` dans le dossier `models/staging/`
2. Copiez-collez le contenu suivant :

```sql
-- Modèle de staging pour les commandes
-- Fichier : models/staging/stg_orders.sql
-- Nettoyage, typage, renommage

{{
    config(
        materialized='view',
        tags=['staging', 'orders']
    )
}}

WITH source AS (
    SELECT * FROM {{ source('staging', 'stg_orders') }}
),

renamed AS (
    SELECT
        -- Clés primaires
        id AS order_id,
        user_id AS customer_id,

        -- Timestamps
        created_at AS order_date,

        -- Métriques
        total_amount AS order_amount,

        -- Attributs
        UPPER(status) AS order_status,
        UPPER(country) AS country_code,
        LOWER(payment_method) AS payment_method,

        -- Métadonnées
        _loaded_at

    FROM source
    WHERE total_amount > 0
)

SELECT * FROM renamed
```

3. Sauvegardez le fichier

8.5 Création des dimensions

9 Dimension : dim_customers.sql

Fichier à créer :
`models/core/dimensions/dim_customers.sql`

1. Créez un nouveau fichier nommé `dim_customers.sql` dans le dossier `models/core/dimensions/`
2. Copiez-collez le contenu suivant :

```
-- Dimension Clients
-- Fichier : models/core/dimensions/dim_customers. sql

{{
    config(
        materialized='table',
        tags=['core', 'dimension']
    )
}}

WITH users AS (
    SELECT * FROM {{ source('staging', 'stg_users') }}
),

orders_agg AS (
    SELECT
        user_id,
        MIN(created_at) AS first_order_date,
        MAX(created_at) AS last_order_date,
        COUNT(DISTINCT id) AS total_orders,
        SUM(total_amount) AS lifetime_value
    FROM {{ source('staging', 'stg_orders') }}
    WHERE status IN ('paid', 'shipped', 'delivered')
    GROUP BY user_id
```

```
),

final AS (
    SELECT
        -- Clé primaire
        u.id AS customer_key,

        -- Attributs démographiques
        u.email,
        u.first_name,
        u.last_name,
        u.first_name || ' ' || u.last_name AS full_name,
        UPPER(u.country) AS country_code,

        -- Segmentation
        CASE
            WHEN u.plan_type IN ('premium', 'enterprise') THEN '
            ELSE 'Freemium'
        END AS customer_segment,

        u.plan_type,
        u.is_active,

        -- Dates importantes
        u.created_at AS registration_date,
        u.last_login,
        o.first_order_date,
        o.last_order_date,

        -- Métriques calculées
        COALESCE(o.total_orders, 0) AS total_orders,
        COALESCE(o.lifetime_value, 0) AS lifetime_value,

        -- Classification RFM simplifié
        CASE
            WHEN DATEDIFF(day, o.last_order_date, CURRENT_DATE()
            WHEN DATEDIFF(day, o.last_order_date, CURRENT_DATE()
            WHEN o.last_order_date IS NOT NULL THEN 'Churned'
            ELSE 'No Purchase'
        END AS customer_status,

        -- Métadonnées
        CURRENT_TIMESTAMP() AS _dbt_updated_at
```

```
    FROM users u
    LEFT JOIN orders_agg o ON u.id = o.user_id
)

SELECT * FROM final
```

3. Sauvegardez le fichier

10 Dimension : dim_products.sql

Fichier à créer :
`models/core/dimensions/dim_products.sql`

1. Créez un nouveau fichier nommé `dim_products.sql` dans le dossier `models/core/dimensions/`
2. Copiez-collez le contenu suivant :

```
-- Dimension Produits
-- Fichier : models/core/dimensions/dim_products. sql

{{
    config(
        materialized='table',
        tags=['core', 'dimension']
    )
}}

WITH products AS (
    SELECT * FROM {{ source('staging', 'stg_products') }}
),

product_stats AS (
    SELECT
        product_id,
        SUM(quantity) AS total_quantity_sold,
        SUM(line_total) AS total_revenue,
        COUNT(DISTINCT order_id) AS total_orders
```

```
    FROM {{ source('staging', 'stg_order_items') }}
    GROUP BY product_id
),

final AS (
    SELECT
        -- Clé primaire
        p.id AS product_key,

        -- Attributs
        p.name AS product_name,
        p. description AS product_description,
        p. category AS product_category,
        p.merchant_id,

        -- Prix
        p.price AS current_price,

        -- Stock
        p.stock_quantity AS current_stock,

        -- Métriques calculées
        COALESCE(s.total_quantity_sold, 0) AS total_quantity_sol
        COALESCE(s.total_revenue, 0) AS total_revenue,
        COALESCE(s.total_orders, 0) AS total_orders,

        -- Classification
        CASE
            WHEN s.total_revenue IS NULL THEN 'No Sales'
            ELSE 'Active'
        END AS product_status,

        -- Dates
        p.created_at AS product_created_at,
        p.updated_at AS product_updated_at,

        -- Métadonnées
        CURRENT_TIMESTAMP() AS _dbt_updated_at

    FROM products p
    LEFT JOIN product_stats s ON p.id = s.product_id
)
```

```
SELECT * FROM final
```

3. Sauvegardez le fichier

8.6 Création de la table de faits

11 Fait : fact_orders.sql

Fichier à créer :
`models/core/facts/fact_orders.sql`

1. Créez un nouveau fichier nommé `fact_orders.sql` dans le dossier `models/core/facts/`
2. Copiez-collez le contenu suivant :

```
-- Table de faits des commandes
-- Fichier : models/core/facts/fact_orders.sql
-- Granularité : ligne de commande

{{
    config(
        materialized='table',
        tags=['core', 'fact']
    )
}}

WITH orders AS (
    SELECT * FROM {{ source('staging', 'stg_orders') }}
),

order_items AS (
    SELECT * FROM {{ source('staging', 'stg_order_items') }}
),

final AS (
```

```
    SELECT
        -- Clés de faits
        oi.id AS order_item_key,
        o.id AS order_key,

        -- Clés étrangères (dimensions)
        o.user_id AS customer_key,
        oi.product_id AS product_key,
        TO_DATE(o.created_at) AS date_key,

        -- Attributs dégénérés
        o.status AS order_status,
        o. country AS country_code,
        o.payment_method,

        -- Timestamps
        o.created_at AS order_timestamp,

        -- Métriques additives
        oi.quantity AS quantity_sold,
        oi.unit_price,
        oi.line_total AS line_revenue,
        o.total_amount AS order_total,

        -- Métriques dérivées
        oi.line_total * 0.2 AS estimated_margin,

        -- Flags
        CASE WHEN o.status IN ('paid', 'shipped', 'delivered') TH
        CASE WHEN o.status = 'cancelled' THEN 1 ELSE 0 END AS is_

    FROM order_items oi
    INNER JOIN orders o ON oi. order_id = o.id
    WHERE o.created_at >= '{{ var("start_date") }}'
)

SELECT * FROM final
```

3. Sauvegardez le fichier

8.7 Création du data mart

12 Data Mart : mart_sales_overview.sql

> **Fichier à créer :**
> `models/marts/mart_sales_overview.sql`

1. Créez un nouveau fichier nommé `mart_sales_overview.sql` dans le dossier `models/marts/`
2. Copiez-collez le contenu suivant :

```
-- Vue d'ensemble des ventes
-- Fichier : models/marts/mart_sales_overview.sql
-- Agrégation par jour, pays, catégorie

{{
    config(
        materialized='table',
        tags=['mart', 'sales']
    )
}}

WITH fact_orders AS (
    SELECT * FROM {{ ref('fact_orders') }}
),

dim_products AS (
    SELECT * FROM {{ ref('dim_products') }}
),

dim_customers AS (
    SELECT * FROM {{ ref('dim_customers') }}
),

daily_sales AS (
    SELECT
        f.date_key AS sale_date,
        f.country_code,
        p.product_category,
        c.customer_segment,

        -- Métriques agrégées
```

```
        COUNT(DISTINCT f.order_key) AS total_orders,
        COUNT(DISTINCT f.customer_key) AS unique_customers,
        SUM(f.quantity_sold) AS total_quantity,
        SUM(f.line_revenue) AS total_revenue,
        SUM(f.estimated_margin) AS total_margin,
        AVG(f.order_total) AS avg_order_value,

        -- Ratios
        SUM(f.line_revenue) / NULLIF(COUNT(DISTINCT f.order_key),
        SUM(f.estimated_margin) / NULLIF(SUM(f.line_revenue), 0)

    FROM fact_orders f
    INNER JOIN dim_products p ON f.product_key = p.product_key
    INNER JOIN dim_customers c ON f.customer_key = c.customer_ke
    WHERE f.is_completed = 1
    GROUP BY 1, 2, 3, 4
)

SELECT * FROM daily_sales
```

3. Sauvegardez le fichier

8.8 Exécution de dbt

13 Exécution des modèles dbt

1. Dans votre invite de commande, assurez-vous d'être dans le dossier dbt :

```
cd ~/ShopStreamTP/shopstream_dbt
```

2. Exécutez tous les modèles :

```
dbt run
```

3. dbt compile et exécute tous les modèles dans l'ordre des dépendances

4. Vous voyez :

```
Running with dbt=1.7.4
Found 6 models, 0 tests, 0 snapshots, 0 analyses, 0 macros, 0

Concurrency: 4 threads (target='dev')

1 of 6 START sql view model CORE.stg_orders .................
1 of 6 OK created sql view model CORE.stg_orders . ..........
2 of 6 START sql table model CORE.dim_customers .............
3 of 6 START sql table model CORE.dim_products ..............
2 of 6 OK created sql table model CORE.dim_customers ........
3 of 6 OK created sql table model CORE.dim_products .........
4 of 6 START sql table model CORE. fact_orders ..............
4 of 6 OK created sql table model CORE.fact_orders ..........
5 of 6 START sql table model MARTS.mart_sales_overview ......
5 of 6 OK created sql table model MARTS.mart_sales_overview .

Completed successfully

Done.  PASS=6 WARN=0 ERROR=0 SKIP=0 TOTAL=6
```

5. Tous les modèles sont créés dans Snowflake

14 Exécution des tests dbt

1. Exécutez les tests de qualité de données :

```
dbt test
```

2. dbt exécute tous les tests définis dans les fichiers YAML
3. Vous voyez :

```
Running with dbt=1.7. 4
Found 6 models, 8 tests, 0 snapshots, 0 analyses, 0 macros, 0

Concurrency: 4 threads (target='dev')
```

```
1 of 8 START test not_null_stg_users_id .....................
2 of 8 START test unique_stg_users_id . .....................
3 of 8 START test not_null_stg_users_email . ................
4 of 8 START test unique_stg_users_email ... ................
1 of 8 PASS not_null_stg_users_id . .... ....................
2 of 8 PASS unique_stg_users_id . ...... ....................
3 of 8 PASS not_null_stg_users_email .. .....................
4 of 8 PASS unique_stg_users_email ..........................
...

Completed successfully

Done. PASS=8 WARN=0 ERROR=0 SKIP=0 TOTAL=8
```

4. Tous les tests passent avec succès

15 Génération de la documentation

1. Générez la documentation :

```
dbt docs generate
```

2. Lancez le serveur de documentation :

```
dbt docs serve
```

3. Votre navigateur s'ouvre automatiquement sur `http://localhost:8080`
4. Vous voyez la documentation complète de votre projet dbt avec :
   - Le lineage graph (graphe des dépendances)
   - La description de chaque modèle
   - Les résultats des tests
   - Le code SQL de chaque modèle

5. Fermez le serveur avec `Ctrl+C` dans le terminal

16 Vérification dans Snowflake

1. Retournez dans Snowflake (interface web)
2. Allez dans Worksheets
3. Exécutez :

```
USE DATABASE SHOPSTREAM_DWH;
USE SCHEMA MARTS;

SELECT * FROM mart_sales_overview
LIMIT 10;
```

4. Vous voyez les données agrégées de ventes
5. Vos transformations dbt sont bien dans Snowflake

Récapitulatif dbt

Vous avez réussi à :

- Installer dbt avec l'adaptateur Snowflake
- Configurer la connexion Snowflake
- Créer des modèles de staging
- Créer des dimensions (dim_customers, dim_products)
- Créer une table de faits (fact_orders)
- Créer un data mart (mart_sales_overview)
- Exécuter les transformations avec dbt run
- Tester la qualité des données avec dbt test
- Générer la documentation avec dbt docs

> **Prochaine étape** : Orchestration avec Apache Airflow

9. Création des Data Marts

Contexte dans le pipeline

Nous restons sur l'étape dbt pour créer des data marts supplémentaires. Ces tables sont optimisées pour répondre à des questions métier spécifiques.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)
Airflow (Orchestration)		Power BI (BI)	

9.1 Data Mart : Performance Produits

1 Création de mart_product_performance.sql

> Fichier à créer :
> `models/marts/mart_product_performance. sql`

1. Créez un nouveau fichier nommé `mart_product_performance.sql` dans le dossier `models/marts/`
2. Copiez-collez le contenu suivant :

```
-- Performance des produits (analyse ABC)
-- Fichier : models/marts/mart_product_performance.sql

{{
    config(
        materialized='table',
```

```
        tags=['mart', 'product']
    )
}}

WITH fact_orders AS (
    SELECT * FROM {{ ref('fact_orders') }}
    WHERE is_completed = 1
),

dim_products AS (
    SELECT * FROM {{ ref('dim_products') }}
),

product_metrics AS (
    SELECT
        f.product_key,
        p.product_name,
        p.product_category,

        -- Volume
        COUNT(DISTINCT f.order_key) AS total_orders,
        SUM(f.quantity_sold) AS total_quantity_sold,

        -- Revenus
        SUM(f. line_revenue) AS total_revenue,
        AVG(f.unit_price) AS avg_unit_price,

        -- Marge
        SUM(f. estimated_margin) AS total_margin,
        SUM(f.estimated_margin) / NULLIF(SUM(f.line_revenue), 0)

        -- Dates
        MIN(f.order_timestamp) AS first_sale_date,
        MAX(f.order_timestamp) AS last_sale_date

    FROM fact_orders f
    INNER JOIN dim_products p ON f.product_key = p.product_key
    GROUP BY 1, 2, 3
),

ranked AS (
    SELECT
        *,
        -- Classement par revenu
```

```
        ROW_NUMBER() OVER (ORDER BY total_revenue DESC) AS reven

        -- Cumul du CA
        SUM(total_revenue) OVER (ORDER BY total_revenue DESC ROW

        -- CA total
        SUM(total_revenue) OVER () AS total_revenue_all

    FROM product_metrics
),

final AS (
    SELECT
        *,
        -- Pourcentage du CA total
        total_revenue / total_revenue_all AS revenue_pct,

        -- Pourcentage cumulé
        cumulative_revenue / total_revenue_all AS cumulative_rev

        -- Classification ABC
        CASE
            WHEN cumulative_revenue / total_revenue_all <= 0.8 TH
            WHEN cumulative_revenue / total_revenue_all <= 0.95 T
            ELSE 'C'
        END AS abc_class,

        -- Performance
        CASE
            WHEN revenue_rank <= 10 THEN 'Top 10'
            WHEN revenue_rank <= 50 THEN 'Top 50'
            ELSE 'Long Tail'
        END AS performance_tier

    FROM ranked
)

SELECT
    product_key,
    product_name,
    product_category,
    total_orders,
    total_quantity_sold,
    total_revenue,
```

```
    avg_unit_price,
    total_margin,
    margin_rate,
    first_sale_date,
    last_sale_date,
    revenue_rank,
    revenue_pct,
    cumulative_revenue_pct,
    abc_class,
    performance_tier
FROM final
ORDER BY revenue_rank
```

3. Sauvegardez le fichier
4. Exécutez dbt pour créer ce nouveau mart :

```
dbt run --select mart_product_performance
```

5. Le mart est créé dans Snowflake

9.2 Data Mart : Customer Lifetime Value

2 Création de mart_customer_ltv.sql

Fichier à créer :
`models/marts/mart_customer_ltv.sql`

1. Créez un nouveau fichier nommé `mart_customer_ltv. sql` dans le dossier `models/marts/`
2. Copiez-collez le contenu suivant :

```
-- Customer Lifetime Value et segmentation RFM
-- Fichier : models/marts/mart_customer_ltv.sql

{{
```

```
    config(
        materialized='table',
        tags=['mart', 'customer']
    )
}}

WITH dim_customers AS (
    SELECT * FROM {{ ref('dim_customers') }}
),

fact_orders AS (
    SELECT * FROM {{ ref('fact_orders') }}
    WHERE is_completed = 1
),

customer_metrics AS (
    SELECT
        f.customer_key,

        -- Récence (Recency)
        MAX(f.order_timestamp) AS last_order_date,
        DATEDIFF(day, MAX(f.order_timestamp), CURRENT_DATE()) AS

        -- Fréquence (Frequency)
        COUNT(DISTINCT f.order_key) AS total_orders,
        DATEDIFF(day, MIN(f.order_timestamp), MAX(f.order_timest

        -- Montant (Monetary)
        SUM(f.line_revenue) AS lifetime_value,
        AVG(f.order_total) AS avg_order_value,

        -- Dates
        MIN(f.order_timestamp) AS first_order_date

    FROM fact_orders f
    GROUP BY 1
),

rfm_scores AS (
    SELECT
        *,
        -- Score RFM (1-5, 5 étant le meilleur)
        NTILE(5) OVER (ORDER BY days_since_last_order ASC) AS re
        NTILE(5) OVER (ORDER BY total_orders DESC) AS frequency_
```

```
        NTILE(5) OVER (ORDER BY lifetime_value DESC) AS monetary_

    FROM customer_metrics
),

rfm_segments AS (
    SELECT
        *,
        recency_score + frequency_score + monetary_score AS rfm_

        -- Segmentation RFM
        CASE
            WHEN recency_score >= 4 AND frequency_score >= 4 AND
            WHEN recency_score >= 3 AND frequency_score >= 3 THEN
            WHEN recency_score >= 4 AND frequency_score <= 2 THEN
            WHEN recency_score >= 3 AND monetary_score >= 3 THEN
            WHEN recency_score <= 2 AND frequency_score >= 3 THEN
            WHEN recency_score <= 2 AND monetary_score >= 4 THEN
            WHEN recency_score <= 2 THEN 'Hibernating'
            ELSE 'Others'
        END AS rfm_segment

    FROM rfm_scores
),

final AS (
    SELECT
        c.customer_key,
        c.email,
        c.full_name,
        c.country_code,
        c.customer_segment,
        c.customer_status,

        -- Métriques RFM
        r.last_order_date,
        r.days_since_last_order,
        r.total_orders,
        r.customer_lifespan_days,
        r.lifetime_value,
        r.avg_order_value,
        r.first_order_date,

        -- Scores et segments
```

```
        r.recency_score,
        r.frequency_score,
        r.monetary_score,
        r.rfm_total_score,
        r.rfm_segment,

        -- Risque de churn
        CASE
            WHEN r.days_since_last_order > 180 THEN 'High'
            WHEN r.days_since_last_order > 90 THEN 'Medium'
            ELSE 'Low'
        END AS churn_risk

    FROM dim_customers c
    INNER JOIN rfm_segments r ON c.customer_key = r. customer_ke
)

SELECT * FROM final
```

3. Sauvegardez le fichier
4. Exécutez dbt :

```
dbt run --select mart_customer_ltv
```

5. Le mart est créé dans Snowflake

Data Marts créés

Vous disposez maintenant de 3 data marts dans Snowflake :

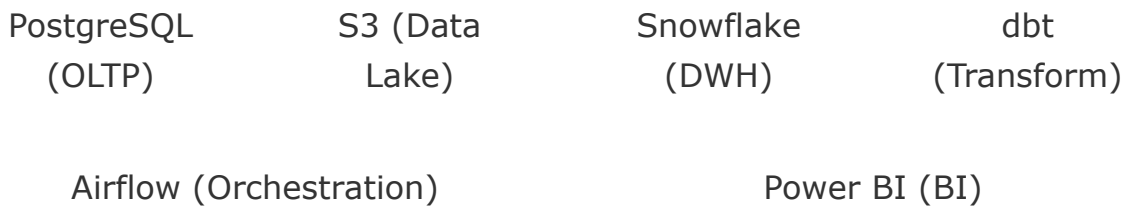
- **mart_sales_overview** : Vue d'ensemble des ventes par jour, pays, catégorie
- **mart_product_performance** : Performance des produits avec analyse ABC
- **mart_customer_ltv** : Lifetime Value et segmentation RFM des clients

> Ces tables sont prêtes à être consommées par Power BI.

10. Installation et configuration d'Apache Airflow

Contexte dans le pipeline

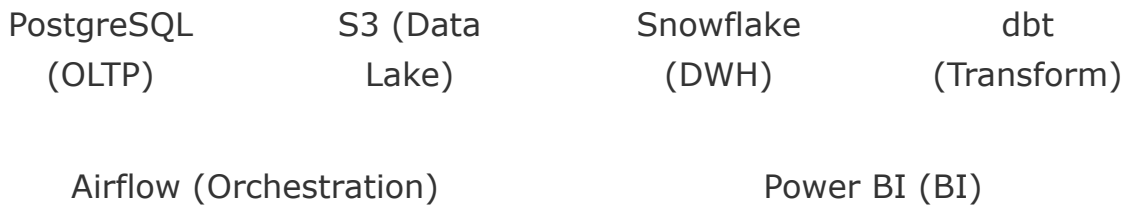
Nous passons maintenant à la cinquième étape : Apache Airflow. Jusqu'ici, nous avons exécuté chaque étape manuellement. Airflow va orchestrer automatiquement l'ensemble du pipeline : extraction PostgreSQL vers S3, chargement S3 vers Snowflake, transformations dbt, et rafraîchissement BI.



10. Installation et configuration d'Apache Airflow

Contexte dans le pipeline

Nous passons maintenant à la cinquième étape : Apache Airflow. Jusqu'ici, nous avons exécuté chaque étape manuellement. Airflow va orchestrer automatiquement l'ensemble du pipeline : extraction PostgreSQL vers S3, chargement S3 vers Snowflake, transformations dbt, et rafraîchissement BI.



10.1 Qu'est-ce qu'Apache Airflow ?

Apache Airflow est une plateforme open-source d'orchestration de workflows. Elle permet de définir, planifier et monitorer des pipelines de données complexes sous forme de **DAGs** (Directed Acyclic Graphs).

Concepts clés d'Airflow :

- **DAG** : Un graphe orienté acyclique qui définit un workflow
- **Task** : Une unité de travail (BashOperator, PythonOperator, SnowflakeOperator)
- **Operator** : Un template de tâche réutilisable
- **Sensor** : Attend qu'une condition soit remplie (ex : fichier présent dans S3)
- **Schedule** : Fréquence d'exécution (expression cron)
- **Executor** : Mécanisme d'exécution (Local, Celery, Kubernetes)

10.2 Installation d'Airflow

1 Installation d'Apache Airflow

1. Ouvrez une invite de commande (CMD ou Terminal)
2. Définissez la version d'Airflow à installer :

```
export AIRFLOW_VERSION=2. 8.0
export PYTHON_VERSION="$(python --version | cut -d "
" -f 2 | cut -d "." -f 1-2)"
export
CONSTRAINT_URL="https://raw.githubusercontent.com/apache/airfl
```

Sous Windows (CMD), utilisez plutôt :

```
set AIRFLOW_VERSION=2.8.0
set
```

```
CONSTRAINT_URL=https://raw.githubusercontent.com/apache/airflo
2.8.0/constraints-3.11.txt
```

3. Installez Airflow avec les providers nécessaires :

```
pip install "apache-
airflow[postgres,amazon,snowflake]==${AIRFLOW_VERSION}"
--constraint "${CONSTRAINT_URL}"
```

Sous Windows :

```
pip install "apache-
airflow[postgres,amazon,snowflake]==2.8.0" --
constraint
"https://raw.githubusercontent.com/apache/airflow/constraints-
2.8.0/constraints-3.11.txt"
```

4. L'installation prend 5-10 minutes
5. Vérifiez l'installation :

```
airflow version
```

6. Vous devez voir : `2.8.0`
7. Airflow est installé

2 Configuration du dossier Airflow

> **Dossier Airflow à créer :**
> Windows : `C:\Users\VotreNom\ShopStreamTP\airflow\`
> macOS/Linux : `~/ShopStreamTP/airflow/`

1. Définissez la variable d'environnement AIRFLOW_HOME :
   Windows (CMD) :

```
set
AIRFLOW_HOME=C:\Users\VotreNom\ShopStreamTP\airflow
```

macOS/Linux :

```
export AIRFLOW_HOME=~/ShopStreamTP/airflow
```

2. Initialisez la base de données Airflow :

```
airflow db init
```

3. La commande crée la structure de dossiers :

```
airflow/
├── airflow. cfg
├── airflow.db
├── dags/
├── logs/
└── plugins/
```

4. Le dossier Airflow est initialisé

3 Création d'un utilisateur admin

1. Créez un utilisateur administrateur :

```
airflow users create \
--username admin \
--firstname Admin \
--lastname User \
--role Admin \
--email admin@shopstream.com \
--password admin123
```

Sous Windows (tout sur une ligne) :

```
airflow users create --username admin --firstname
Admin --lastname User --role Admin --email
admin@shopstream.com --password admin123
```

2. Vous voyez : `User "admin" created`
3. L'utilisateur admin est créé

4 Démarrage d'Airflow

1. Ouvrez **deux** invites de commande (ou deux onglets Terminal)
2. **Terminal 1** : Démarrez le scheduler :

```
airflow scheduler
```

3. Le scheduler démarre. Laissez ce terminal ouvert
4. **Terminal 2** : Démarrez le webserver :

```
airflow webserver --port 8080
```

5. Le webserver démarre (peut prendre 30 secondes)
6. Vous voyez : `Listening at: http://0.0.0.0:8080`
7. Ouvrez votre navigateur et allez sur : `http://localhost:8080`
8. La page de connexion Airflow s'affiche
9. Connectez-vous avec :
   - Username : `admin`
   - Password : `admin123`
10. Vous voyez le tableau de bord Airflow
11. Airflow est opérationnel

10.3 Configuration des connexions Airflow

5 Configuration de la connexion AWS

1. Dans l'interface Airflow (http://localhost:8080)
2. Allez dans le menu : **Admin > Connections**
3. Cliquez sur le bouton **"+"** (Add a new record)
4. Remplissez le formulaire :
   - **Connection Id** : `aws_default`
   - **Connection Type** : `Amazon Web Services`
   - **AWS Access Key ID** : Votre Access Key ID
   - **AWS Secret Access Key** : Votre Secret Access Key
   - **Extra** : `{"region_name": "eu-west-3"}`
5. Cliquez sur **"Save"**
6. La connexion AWS est configurée

6 Configuration de la connexion Snowflake

1. Dans Airflow, allez dans **Admin > Connections**
2. Cliquez sur le bouton **"+"**
3. Remplissez le formulaire :
   - **Connection Id** : `snowflake_default`
   - **Connection Type** : `Snowflake`
   - **Account** : Votre account Snowflake (ex : `abc12345.eu-west-3.aws`)
   - **User** : Votre username Snowflake

- **Password** : Votre mot de passe Snowflake
- **Database** : `SHOPSTREAM_DWH`
- **Schema** : `STAGING`
- **Warehouse** : `LOADING_WH`
- **Role** : `ACCOUNTADMIN`

4. Cliquez sur **"Save"**
5. La connexion Snowflake est configurée

7 Configuration de la connexion PostgreSQL

1. Dans Airflow, allez dans **Admin > Connections**
2. Cliquez sur le bouton **"+"**
3. Remplissez le formulaire :
   - **Connection Id** : `postgres_shopstream`
   - **Connection Type** : `Postgres`
   - **Host** : `localhost`
   - **Database** : `shopstream`
   - **Login** : `postgres`
   - **Password** : Votre mot de passe PostgreSQL
   - **Port** : `5432`
4. Cliquez sur **"Save"**
5. La connexion PostgreSQL est configurée

10.4 Création du DAG Airflow

8 Création du fichier DAG

Fichier à créer :

Windows :

`C:\Users\VotreNom\ShopStreamTP\airflow\dags\shopstream_daily_pip`

macOS/Linux :

`~/ShopStreamTP/airflow/dags/shopstream_daily_pipeline.py`

1. Créez un nouveau fichier nommé `shopstream_daily_pipeline.py` dans le dossier `airflow/dags/`
2. Copiez-collez le contenu suivant :

```
"""
shopstream_daily_pipeline.py
DAG Airflow pour le pipeline quotidien ShopStream

Ce DAG orchestre :
1. Extraction PostgreSQL vers S3
2. Chargement S3 vers Snowflake Staging
3. Transformations dbt (staging vers core vers marts)
4. Tests de qualité dbt

Emplacement : airflow/dags/shopstream_daily_pipeline.py
Auteur : Data Engineering Team
Date : 2025-11-27
"""

from datetime import datetime, timedelta
from airflow import DAG
from airflow.operators. python import PythonOperator
from airflow.operators.bash import BashOperator
from airflow.providers.amazon.aws.sensors.s3 import S3KeySensor
from airflow.providers.snowflake.operators.snowflake import Snow

# Configuration du DAG
default_args = {
    'owner': 'data_engineering',
    'depends_on_past': False,
```

```
    'start_date': datetime(2025, 11, 1),
    'email': ['alerts@shopstream.com'],
    'email_on_failure': True,
    'email_on_retry': False,
    'retries': 2,
    'retry_delay': timedelta(minutes=5),
    'execution_timeout': timedelta(hours=2)
}

dag = DAG(
    'shopstream_daily_pipeline',
    default_args=default_args,
    description='Pipeline quotidien ShopStream : PostgreSQL vers
    schedule_interval='0 2 * * *',  # Tous les jours à 2h du mat
    catchup=False,
    tags=['production', 'daily', 'shopstream']
)

# Tâches Python
def extract_postgres_to_s3(**context):
    """Extrait les données PostgreSQL et les pousse dans S3"""
    import subprocess
    execution_date = context['ds']  # Date d'exécution (YYYY-MM-
    print(f"Extraction PostgreSQL vers S3 pour {execution_date}"

    # Exécution du script Python d'export
    # MODIFIEZ LE CHEMIN SELON VOTRE INSTALLATION
    result = subprocess.run(
        ['python', '/chemin/vers/ShopStreamTP/scripts/export_to_
        capture_output=True,
        text=True
    )

    if result.returncode != 0:
        raise Exception(f"Erreur lors de l'export : {result.stde

    print(result.stdout)
    print("Export terminé avec succès")

def run_dbt_models(**context):
    """Exécute les transformations dbt"""
    import subprocess
```

```
    print("Exécution des modèles dbt")

    # MODIFIEZ LE CHEMIN SELON VOTRE INSTALLATION
    result = subprocess.run(
        ['dbt', 'run', '--project-dir', '/chemin/vers/ShopStream'
        capture_output=True,
        text=True
    )

    if result.returncode != 0:
        raise Exception(f"Erreur dbt run : {result.stderr}")

    print(result.stdout)
    print("Transformations dbt terminées")

def run_dbt_tests(**context):
    """Exécute les tests de qualité dbt"""
    import subprocess

    print("Exécution des tests dbt")

    # MODIFIEZ LE CHEMIN SELON VOTRE INSTALLATION
    result = subprocess.run(
        ['dbt', 'test', '--project-dir', '/chemin/vers/ShopStrea
        capture_output=True,
        text=True
    )

    if result.returncode != 0:
        print(f"Certains tests ont échoué : {result.stderr}")
        # On ne fait pas échouer le DAG, juste un warning

    print(result.stdout)
    print("Tests dbt terminés")

def send_success_notification(**context):
    """Envoie une notification de succès"""
    print("Pipeline terminé avec succès")
    print(f"Execution date : {context['ds']}")
    # Ici : appel à Slack/Teams/Email via webhook

# Définition des tâches

# Tâche 1 : Extraction PostgreSQL vers S3
```

```
task_extract = PythonOperator(
    task_id='extract_postgres_to_s3',
    python_callable=extract_postgres_to_s3,
    dag=dag
)

# Tâche 2 : Attendre que les fichiers soient présents dans S3
task_wait_s3_users = S3KeySensor(
    task_id='wait_s3_users_file',
    bucket_name='shopstream-datalake-votreprenom',  # MODIFIEZ a
    bucket_key='raw/postgres/users/{{ ds }}/users_{{ ds_nodash }
    aws_conn_id='aws_default',
    timeout=600,
    poke_interval=30,
    dag=dag
)

task_wait_s3_orders = S3KeySensor(
    task_id='wait_s3_orders_file',
    bucket_name='shopstream-datalake-votreprenom',  # MODIFIEZ a
    bucket_key='raw/postgres/orders/{{ ds }}/orders_{{ ds_nodash
    aws_conn_id='aws_default',
    timeout=600,
    poke_interval=30,
    dag=dag
)

# Tâche 3 : Chargement S3 vers Snowflake STAGING
task_load_staging_users = SnowflakeOperator(
    task_id='load_staging_users',
    snowflake_conn_id='snowflake_default',
    sql="""
        USE WAREHOUSE LOADING_WH;
        USE SCHEMA SHOPSTREAM_DWH. STAGING;

        TRUNCATE TABLE stg_users;

        COPY INTO stg_users (id, email, first_name, last_name, c
        FROM @RAW. s3_raw_stage/postgres/users/{{ ds }}/
        FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY =
        ON_ERROR = 'CONTINUE';
    """,
    dag=dag
)
```

```
task_load_staging_orders = SnowflakeOperator(
    task_id='load_staging_orders',
    snowflake_conn_id='snowflake_default',
    sql="""
        USE WAREHOUSE LOADING_WH;
        USE SCHEMA SHOPSTREAM_DWH. STAGING;

        TRUNCATE TABLE stg_orders;

        COPY INTO stg_orders (id, user_id, created_at, total_amo
        FROM @RAW.s3_raw_stage/postgres/orders/{{ ds }}/
        FILE_FORMAT = (TYPE = CSV FIELD_OPTIONALLY_ENCLOSED_BY =
        ON_ERROR = 'CONTINUE';
    """,
    dag=dag
)

# Tâche 4 : Exécution de dbt (staging vers core vers marts)
task_dbt_run = PythonOperator(
    task_id='dbt_run_models',
    python_callable=run_dbt_models,
    dag=dag
)

# Tâche 5 : Tests dbt
task_dbt_test = PythonOperator(
    task_id='dbt_test_models',
    python_callable=run_dbt_tests,
    dag=dag
)

# Tâche 6 : Génération de la documentation dbt
task_dbt_docs = BashOperator(
    task_id='dbt_generate_docs',
    bash_command='cd /chemin/vers/ShopStreamTP/shopstream_dbt &&
    dag=dag
)

# Tâche 7 : Notification de succès
task_success = PythonOperator(
    task_id='send_success_notification',
    python_callable=send_success_notification,
    dag=dag
```

```
)

# Définition des dépendances (DAG)

# Phase 1 : Extraction
task_extract >> [task_wait_s3_users, task_wait_s3_orders]

# Phase 2 : Chargement Staging
task_wait_s3_users >> task_load_staging_users
task_wait_s3_orders >> task_load_staging_orders

# Phase 3 : Transformations dbt
[task_load_staging_users, task_load_staging_orders] >> task_dbt_
```

```
# Phase 4 : Tests et documentation
task_dbt_run >> task_dbt_test
task_dbt_test >> task_dbt_docs

# Phase 5 : Notification
task_dbt_docs >> task_success
```

3. Sauvegardez le fichier
4. **IMPORTANT** : Modifiez les lignes suivantes :
   - Ligne 67 et 78 : Chemin vers `export_to_s3.py` et projet dbt
   - Ligne 109 et 121 : Nom de votre bucket S3
   - Ligne 190 : Chemin vers le projet dbt

9 Activation et exécution du DAG

1. Retournez dans l'interface Airflow (http://localhost:8080)
2. Rafraîchissez la page
3. Vous devez voir le DAG **shopstream_daily_pipeline** dans la liste
4. Par défaut, le DAG est en pause (toggle désactivé)
5. Cliquez sur le toggle à gauche du nom du DAG pour l'activer

6. Le DAG est maintenant actif (toggle bleu)
7. Pour tester le DAG manuellement :
    - Cliquez sur le nom du DAG **shopstream_daily_pipeline**
    - Vous voyez la vue détaillée du DAG avec le graphe
    - Cliquez sur le bouton **"Play"** (triangle) en haut à droite
    - Sélectionnez **"Trigger DAG"**
8. Le DAG démarre
9. Vous voyez les tâches s'exécuter une par une :
    - Vert : Succès
    - Jaune : En cours
    - Rouge : Échec
10. Cliquez sur une tâche pour voir les logs détaillés
11. Le pipeline s'exécute automatiquement

10.5 Visualisation du DAG

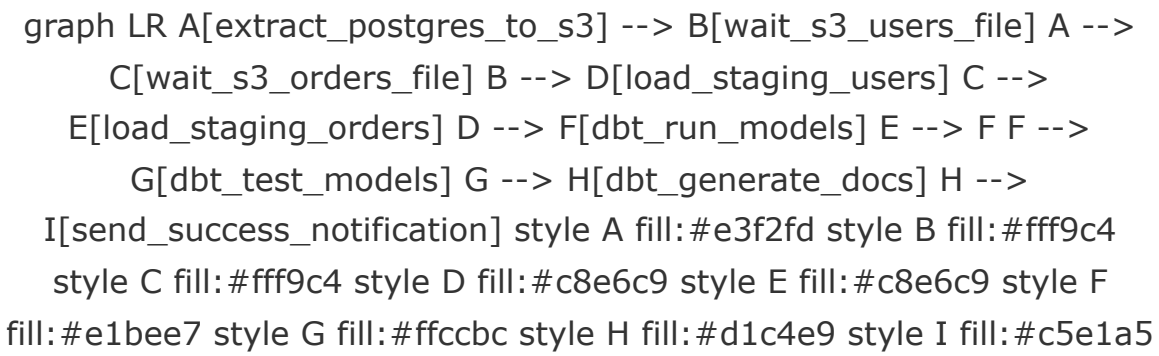


Figure 3 : DAG Airflow du pipeline quotidien

Récapitulatif Airflow

> Vous avez réussi à :
>
> - Installer Apache Airflow avec les providers nécessaires
> - Configurer les connexions (AWS, Snowflake, PostgreSQL)
> - Créer un DAG complet orchestrant tout le pipeline
> - Activer et exécuter le DAG
> - Surveiller l'exécution via l'interface web
>
> **Prochaine étape** : Visualisation avec Power BI

11. Connexion Power BI et création de dashboards

Contexte dans le pipeline

Nous arrivons à la dernière étape technique : Power BI. Les données sont maintenant transformées et disponibles dans les data marts de Snowflake. Nous allons les connecter à Power BI pour créer des dashboards interactifs destinés aux utilisateurs métier.

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)
Airflow (Orchestration)		Power BI (BI)	

11.1 Pourquoi Power BI ?

Power BI est l'outil de BI de Microsoft, largement adopté en entreprise. Il offre une intégration native avec l'écosystème Microsoft et une courbe d'apprentissage douce pour les utilisateurs métier.

Alternatives :

Outil	Avantages	Inconvénients
Power BI	Intégration Microsoft, prix attractif, DAX puissant	Moins flexible que Tableau
Tableau	Visualisations riches, communauté active	Coût élevé
Looker	LookML (code-first), excellent lineage	Nécessite des compétences techniques
Metabase	Open-source, simple, gratuit	Moins de fonctionnalités avancées

11.2 Installation de Power BI Desktop

1 Téléchargement de Power BI Desktop

1. Allez sur https://powerbi.microsoft. com/fr-fr/downloads/
2. Cliquez sur **"Télécharger gratuitement"** sous "Power BI Desktop"
3. Le fichier `PBIDesktopSetup_x64.exe` se télécharge (environ 500 Mo)
4. Double-cliquez sur le fichier téléchargé
5. L'assistant d'installation s'ouvre
6. Cliquez sur **"Next"**
7. Acceptez les conditions d'utilisation
8. Cliquez sur **"Next"**
9. Laissez le dossier d'installation par défaut
10. Cliquez sur **"Install"**

11. L'installation prend 2-3 minutes
12. Cliquez sur **"Finish"**
13. Power BI Desktop s'ouvre automatiquement

> **Note** : Power BI Desktop est disponible uniquement pour Windows. Les utilisateurs Mac peuvent utiliser Power BI Service (version web) ou une machine virtuelle Windows.

11.3 Connexion à Snowflake

2 Ajout de la source de données Snowflake

1. Dans Power BI Desktop, fermez la fenêtre de démarrage si elle apparaît
2. Dans le ruban supérieur, allez dans l'onglet **"Accueil"**
3. Cliquez sur **"Obtenir des données"**
4. Dans la fenêtre qui s'ouvre, tapez `Snowflake` dans la barre de recherche
5. Sélectionnez **"Snowflake"**
6. Cliquez sur **"Connecter"**
7. Une fenêtre "Snowflake" s'ouvre
8. Remplissez les champs :
   - **Serveur** : Votre URL Snowflake (ex : `abc12345.eu-west-3.aws.snowflakecomputing.com`)
   - **Warehouse** : `BI_WH`
9. Laissez les autres options par défaut

10. Cliquez sur **"OK"**
11. Une fenêtre d'authentification apparaît
12. Sélectionnez l'onglet **"Base de données"**
13. Entrez :
    - **Nom d'utilisateur** : Votre username Snowflake
    - **Mot de passe** : Votre mot de passe Snowflake
14. Cliquez sur **"Connecter"**
15. Power BI se connecte à Snowflake (peut prendre 10-20 secondes)
16. La fenêtre "Navigateur" s'ouvre avec la liste des bases de données

3 Sélection des tables Data Marts

1. Dans la fenêtre "Navigateur", développez l'arborescence :
    - Cliquez sur le triangle à côté de **"SHOPSTREAM_DWH"**
    - Cliquez sur le triangle à côté de **"MARTS"**
2. Vous voyez les 3 tables de data marts :
    - `MART_SALES_OVERVIEW`
    - `MART_PRODUCT_PERFORMANCE`
    - `MART_CUSTOMER_LTV`
3. Cochez les 3 tables
4. En bas de la fenêtre, vous avez deux options :
    - **Charger** : Importe les données dans Power BI (mode Import)
    - **Transformer les données** : Ouvre Power Query Editor
5. Cliquez sur **"Charger"**

6. Power BI charge les données (durée : 30 secondes à 2 minutes selon le volume)
7. Vous voyez les 3 tables apparaître dans le panneau "Champs" à droite
8. Les données sont importées dans Power BI

11.4 Création du dashboard "Vue d'ensemble des ventes"

4 Création de la page de rapport

1. Vous êtes maintenant dans la vue "Rapport" de Power BI
2. En bas, vous voyez "Page 1"
3. Double-cliquez sur "Page 1" et renommez-la en **"Vue d'ensemble"**
4. Appuyez sur Entrée

5 Ajout d'une carte KPI : Chiffre d'affaires total

1. Dans le panneau "Visualisations" (à droite),

   cliquez sur l'icône **"Carte"**
2. Un visuel de carte vide apparaît sur le canevas
3. Dans le panneau "Champs" (à droite), développez **MART_SALES_OVERVIEW**
4. Cochez le champ **TOTAL_REVENUE**
5. Le montant total du CA s'affiche dans la carte

6. Redimensionnez la carte en haut à gauche du canevas
7. Pour formater le montant :
   - Sélectionnez la carte
   - Dans le panneau "Visualisations", allez dans l'onglet "Format"
   - Développez **"Étiquettes de données"**
   - Changez le format d'affichage en **"€"** avec 0 décimale
8. Le KPI CA total est créé

6 Ajout d'une carte KPI : Nombre de commandes

1. Cliquez sur une zone vide du canevas pour désélectionner la carte précédente
2. Dans "Visualisations",

   cliquez sur **"Carte"**
3. Dans "Champs", sous MART_SALES_OVERVIEW,

   cochez **TOTAL_ORDERS**
4. Positionnez cette carte à côté de la première
5. Le KPI nombre de commandes est créé

7 Ajout d'un graphique en courbes : Évolution du CA

1. Cliquez sur une zone vide du canevas
2. Dans "Visualisations",

   cliquez sur **"Graphique en courbes"**

3. Dans "Champs", sous MART_SALES_OVERVIEW :
   - Glissez-déposez **SALE_DATE** dans **"Axe X"**
   - Glissez-déposez **TOTAL_REVENUE** dans **"Axe Y"**
4. Le graphique d'évolution du CA s'affiche
5. Redimensionnez le graphique pour qu'il prenne toute la largeur sous les KPI
6. Le graphique d'évolution est créé

8 Ajout d'un graphique en barres : CA par pays

1. Cliquez sur une zone vide du canevas
2. Dans "Visualisations",

   > cliquez sur **"Graphique à barres empilées"**

3. Dans "Champs", sous MART_SALES_OVERVIEW :
   - Glissez-déposez **COUNTRY_CODE** dans **"Axe Y"**
   - Glissez-déposez **TOTAL_REVENUE** dans **"Axe X"**
4. Le graphique CA par pays s'affiche
5. Positionnez le graphique en bas à gauche
6. Le graphique par pays est créé

9 Ajout d'un graphique en secteurs : CA par catégorie

1. Cliquez sur une zone vide du canevas
2. Dans "Visualisations",

   > cliquez sur **"Graphique en secteurs"**

3. Dans "Champs", sous MART_SALES_OVERVIEW :
    - Glissez-déposez **PRODUCT_CATEGORY** dans **"Légende"**
    - Glissez-déposez **TOTAL_REVENUE** dans **"Valeurs"**
4. Le graphique CA par catégorie s'affiche
5. Positionnez le graphique en bas à droite
6. Le graphique par catégorie est créé

11.5 Création de mesures DAX

10 Création d'une mesure : Panier moyen

1. Dans le panneau "Champs",

> faites un clic droit sur **MART_SALES_OVERVIEW**

2. Sélectionnez **"Nouvelle mesure"**
3. Dans la barre de formule en haut, tapez :

```
Panier Moyen = DIVIDE(SUM(MART_SALES_OVERVIEW[TOTAL_REVENUE])
```

4. Appuyez sur Entrée
5. La mesure "Panier Moyen" apparaît dans la liste des champs
6. Créez une nouvelle carte KPI avec cette mesure
7. La mesure Panier Moyen est créée

11 Création d'une mesure : CA du mois précédent

1. Créez une nouvelle mesure sur MART_SALES_OVERVIEW

2. Tapez :

```
CA Mois Précédent =
CALCULATE(
    SUM(MART_SALES_OVERVIEW[TOTAL_REVENUE]),
    DATEADD(MART_SALES_OVERVIEW[SALE_DATE], -1, MONTH)
)
```

3. Appuyez sur Entrée
4. La mesure est créée

12 Création d'une mesure : Croissance mensuelle

1. Créez une nouvelle mesure sur MART_SALES_OVERVIEW
2. Tapez :

```
Croissance Mensuelle =
DIVIDE(
    SUM(MART_SALES_OVERVIEW[TOTAL_REVENUE]) - [CA Mois Précéd
    [CA Mois Précédent],
    0
)
```

3. Appuyez sur Entrée
4. Formatez la mesure en pourcentage (clic droit > Format > Pourcentage)
5. La mesure Croissance Mensuelle est créée

11.6 Création d'une deuxième page : Performance Produits

13 Création de la page "Performance Produits"

1. En bas de l'écran,

   > cliquez sur le bouton **"+"** pour ajouter une nouvelle page

2. Renommez la page en **"Performance Produits"**
3. Créez un tableau avec MART_PRODUCT_PERFORMANCE :
   - Dans "Visualisations",

     > cliquez sur **"Tableau"**

   - Ajoutez les colonnes :
     - PRODUCT_NAME
     - PRODUCT_CATEGORY
     - TOTAL_REVENUE
     - TOTAL_ORDERS
     - ABC_CLASS
     - PERFORMANCE_TIER
4. Redimensionnez le tableau pour qu'il prenne toute la page
5. Triez le tableau par TOTAL_REVENUE décroissant (cliquez sur l'en-tête de colonne)
6. Le tableau des performances produits est créé

11.7 Création d'une troisième page : Analyse Clients

14 Création de la page "Analyse Clients"

1. Ajoutez une nouvelle page et renommez-la en **"Analyse Clients"**
2. Créez un graphique en anneau avec MART_CUSTOMER_LTV :

- Dans "Visualisations",

  > cliquez sur **"Graphique en anneau"**

- Légende : RFM_SEGMENT
- Valeurs : Count of CUSTOMER_KEY

3. Positionnez le graphique en haut à gauche
4. Créez un graphique en barres:
   - Dans "Visualisations",

     > cliquez sur **"Graphique à barres empilées"**

   - Axe Y : RFM_SEGMENT
   - Axe X : LIFETIME_VALUE (somme)
5. Positionnez le graphique en haut à droite
6. Créez un tableau avec les top clients :
   - Dans "Visualisations",

     > cliquez sur **"Tableau"**

   - Ajoutez les colonnes :
     - FULL_NAME
     - EMAIL
     - COUNTRY_CODE
     - LIFETIME_VALUE
     - TOTAL_ORDERS
     - RFM_SEGMENT
     - CHURN_RISK
7. Positionnez le tableau en bas
8. Triez par LIFETIME_VALUE décroissant
9. Le tableau d'analyse clients est créé

11.8 Ajout de filtres interactifs

15 Ajout d'un filtre de date

1. Retournez sur la page "Vue d'ensemble"
2. Dans le panneau "Visualisations",

   > cliquez sur **"Segment"**

3. Dans "Champs", sous MART_SALES_OVERVIEW,

   > cochez **SALE_DATE**

4. Un segment de date apparaît
5. Positionnez-le en haut de la page
6. Pour le rendre interactif :
   - Sélectionnez le segment
   - Dans "Visualisations", allez dans l'onglet "Format"
   - Développez **"Style du segment"**
   - Sélectionnez **"Entre"** (permet de choisir une plage de dates)
7. Le filtre de date est créé
8. Testez-le en sélectionnant une plage de dates : tous les visuels se mettent à jour automatiquement

16 Ajout d'un filtre de pays

1. Cliquez sur une zone vide
2. Dans "Visualisations",

   > cliquez sur **"Segment"**

3. Dans "Champs",

cochez **COUNTRY_CODE**

4. Positionnez le segment à côté du filtre de date
5. Pour afficher les cases à cocher :
   - Sélectionnez le segment
   - Dans "Format", sous "Style du segment"
   - Sélectionnez **"Liste déroulante"**
   - Activez **"Sélection multiple"**
6. Le filtre de pays est créé

11.9 Ajout de visuels conditionnels

17 Mise en forme conditionnelle sur le tableau

1. Allez sur la page "Analyse Clients"
2. Sélectionnez le tableau des clients
3. Dans le panneau "Visualisations", allez dans l'onglet "Format"
4. Développez **"Valeurs de cellule"**
5. Trouvez la colonne **CHURN_RISK**
6. Cliquez sur **"Couleur d'arrière-plan"**
7. Activez **"Mise en forme conditionnelle"**
8. Configurez :
   - Format : **Règles**
   - Règle 1 : Si la valeur est "High", couleur rouge clair
   - Règle 2 : Si la valeur est "Medium", couleur orange clair
   - Règle 3 : Si la valeur est "Low", couleur vert clair
9. Cliquez sur **"OK"**

10. Les cellules de risque de churn sont maintenant colorées selon le niveau de risque

11.10 Publication et partage

18 Enregistrement du rapport

1. Dans le menu **"Fichier"**,

   > cliquez sur **"Enregistrer"**

2. Choisissez un emplacement (par exemple : `C:\Users\VotreNom\ShopStreamTP\powerbi\`)
3. Nom du fichier : `ShopStream_Dashboard. pbix`
4. > Cliquez sur **"Enregistrer"**
5. Le rapport est sauvegardé

19 Publication sur Power BI Service (optionnel)

> **Prérequis** : Un compte Microsoft 365 ou un compte Power BI Pro/Premium

1. Dans Power BI Desktop,

   > cliquez sur **"Publier"** dans le ruban "Accueil"

2. Connectez-vous avec votre compte Microsoft si demandé
3. Sélectionnez un espace de travail (par exemple : "Mon espace de travail")

4. Cliquez sur **"Sélectionner"**
5. Power BI publie le rapport (durée : 1-2 minutes)
6. Vous voyez : "Réussite - ShopStream_Dashboard.pbix a été publié"
7. Cliquez sur **"Ouvrir dans Power BI"**
8. Votre navigateur s'ouvre sur Power BI Service
9. Vous pouvez maintenant partager le rapport avec d'autres utilisateurs

11.11 Configuration du rafraîchissement automatique

20 Planification du rafraîchissement (Power BI Service)

1. Dans Power BI Service, allez dans votre espace de travail
2. Trouvez le dataset **"ShopStream_Dashboard"**
3. Cliquez sur les trois points à droite du dataset
4. Sélectionnez **"Paramètres"**
5. Développez **"Actualisation planifiée"**
6. Activez **"Conserver vos données à jour"**
7. Configurez la fréquence :
   - Fréquence d'actualisation : **Quotidienne**
   - Heure : **08:00** (après l'exécution du DAG Airflow à 2h)
8. Activez **"Envoyer une notification d'échec d'actualisation"**
9. Cliquez sur **"Appliquer"**

10. Le rafraîchissement automatique est configuré
11. Les données seront mises à jour automatiquement chaque matin

Récapitulatif Power BI

Vous avez réussi à :

- Installer Power BI Desktop
- Connecter Power BI à Snowflake
- Importer les tables data marts
- Créer 3 pages de dashboards interactifs :
  - Vue d'ensemble des ventes (KPI, évolutions, répartitions)
  - Performance produits (tableau avec classification ABC)
  - Analyse clients (segmentation RFM, CLV, risque de churn)
- Créer des mesures DAX calculées
- Ajouter des filtres interactifs
- Configurer la mise en forme conditionnelle
- Publier et planifier le rafraîchissement automatique

Le pipeline complet est maintenant opérationnel

12. Travail à rendre et livrables

Récapitulatif complet du pipeline

Vous avez construit un pipeline de données moderne de bout en bout :

PostgreSQL (OLTP)	S3 (Data Lake)	Snowflake (DWH)	dbt (Transform)

Airflow (Orchestration) Power BI (BI)

Chaque étape est automatisée et orchestrée. Les données circulent de la source OLTP jusqu'aux dashboards décisionnels en passant par toutes les transformations nécessaires.

12.1 Objectifs pédagogiques atteints

À la fin de ce TP, vous devez être capable de :

- Expliquer l'architecture d'un Modern Data Stack et le rôle de chaque composant
- Concevoir un schéma OLTP normalisé (PostgreSQL)
- Organiser un Data Lake dans S3 (partitionnement, formats)
- Charger des données dans Snowflake avec COPY INTO
- Écrire des modèles dbt (staging, dimensions, faits, marts)
- Orchestrer un pipeline avec Apache Airflow (DAG)
- Créer des dashboards BI connectés à Snowflake
- Argumenter les choix technologiques (OLTP vs OLAP, cloud vs on-premise)

12.2 Questions de réflexion

Question 1 : Architecture et justification des choix

Consigne :

Dessinez l'architecture complète du pipeline ShopStream (vous pouvez utiliser draw.io, Lucidchart, ou un outil similaire). Pour chaque composant, expliquez :

- Son rôle dans le pipeline
- Pourquoi ce choix plutôt qu'une alternative (exemple : pourquoi Snowflake plutôt que PostgreSQL pour l'analytique ?)

- Les flux de données entre composants (format, fréquence, volume estimé)

Éléments à inclure dans votre réponse :

- Diagramme d'architecture avec toutes les couches
- Tableau comparatif justifiant chaque choix technologique
- Estimation des coûts mensuels du pipeline en production (calculez le coût de chaque service)

Question 2 : Comparaison avec une architecture on-premise

Consigne :

Comparez l'architecture moderne (cloud) construite dans ce TP avec une architecture classique on-premise utilisant des outils comme Pentaho, Talend, ou Oracle Data Integrator.

Remplissez ce tableau comparatif :

Critère	Architecture On-Premise (Pentaho)	Architecture Cloud (ce TP)	Avantage pour
Coût initial			
Coût opérationnel (3 ans)			
Scalabilité			
Maintenance			
Performance sur gros			

volumes			
Compétences requises			
Time-to-market			
Sécurité des données			

Conclusion : Dans quels contextes recommanderiez-vous chaque architecture ?

Question 3 : Proposition de nouveaux KPIs

Consigne :

Proposez 3 nouveaux KPIs métier pertinents pour ShopStream qui ne sont pas encore implémentés dans le pipeline.

Pour chaque KPI, fournissez :

1. **Nom et définition** : Quel est le KPI et que mesure-t-il ?
2. **Formule de calcul** : Comment le calculer précisément ?
3. **Intérêt métier** : Pourquoi ce KPI est-il important ? Quelle décision aide-t-il à prendre ?
4. **Sources de données** : Quelles tables du pipeline faut-il utiliser ?
5. **Requête SQL ou modèle dbt** : Écrivez le code SQL pour calculer ce KPI
6. **Visualisation recommandée** : Quel type de graphique Power BI utiliser ?

Exemples de KPIs possibles (n'utilisez pas ceux-ci, proposez les vôtres) :

- Taux de réachat à 30 jours

- Coût d'acquisition client (CAC)
- Ratio LTV/CAC
- Taux d'abandon de panier
- Délai moyen entre inscription et premier achat

Question 4 : Plan de tests et qualité de données

Consigne :

Rédigez un plan de tests complet (2-3 pages) pour garantir la qualité du pipeline ShopStream.

Votre plan doit inclure :

A. Tests de données (Data Quality)

- Liste des règles métier à valider (exemples : email valide, montants positifs, dates cohérentes)
- Pour chaque règle :
  - Table concernée
  - Colonne(s) testée(s)
  - Type de test (unicité, non-nullité, plage de valeurs, format)
  - Requête SQL de test
  - Seuil d'alerte (exemple : si plus de 1% d'erreurs, alerter)

B. Tests de transformations

- Comment vérifier que les jointures dbt sont correctes ?
- Comment valider que les agrégations sont justes ?
- Comment tester la logique de calcul des KPIs ?
- Proposez au moins 3 tests unitaires dbt avec le code YAML

C. Tests de bout en bout

- Décrivez un scénario de test E2E complet : de l'insertion d'une ligne dans PostgreSQL jusqu'à sa visualisation dans Power BI
- Comment automatiser ce test ?
- Combien de temps doit prendre ce test ?

D. Tests de régression

- Comment s'assurer qu'une modification du code dbt ne casse pas les dashboards existants ?
- Proposez une stratégie de versioning et de déploiement (dev, staging, prod)

Question 5 : Sécurité et conformité RGPD

Consigne :

ShopStream opère en Europe et doit respecter le RGPD (Règlement Général sur la Protection des Données).

Répondez aux questions suivantes :

A. Identification des données personnelles

- Quelles données du pipeline ShopStream sont considérées comme "données personnelles" au sens du RGPD ?
- Classez-les par sensibilité (normale, sensible, très sensible)

B. Droit à l'oubli

- Un utilisateur demande la suppression complète de ses données. Décrivez la procédure technique étape par étape pour honorer cette demande dans le pipeline
- Quelles tables faut-il modifier dans PostgreSQL ?
- Comment propager cette suppression dans Snowflake ?
- Que faire des données historiques dans S3 ?
- Écrivez le script SQL pour anonymiser un utilisateur spécifique

C. Anonymisation et pseudonymisation

- Proposez une stratégie pour anonymiser les emails dans le data warehouse tout en gardant la possibilité de faire des analyses de cohortes
- Écrivez une fonction dbt pour hasher les emails de manière déterministe
- Comment implémenter la pseudonymisation dans le pipeline ?

D. Contrôle d'accès (Row-Level Security)

- Comment implémenter le RLS dans Snowflake pour que :
  - L'équipe marketing ne voit que les données européennes
  - Les analystes juniors ne voient pas les données financières sensibles
  - Chaque marchand ne voit que ses propres données
- Écrivez les commandes SQL Snowflake pour implémenter ces règles

E. Audit et traçabilité

- Comment logger qui accède à quelles données et quand ?
- Où stocker ces logs d'audit ?
- Combien de temps les conserver ?
- Comment automatiser la génération de rapports d'audit pour la CNIL ?

12.3 Livrables attendus

Livrable 1 : Document de synthèse (PDF)

Format : PDF de 15-20 pages

Contenu :

- Page de garde (titre, auteurs, date)
- Sommaire
- Introduction : contexte ShopStream, objectifs du pipeline
- Architecture détaillée avec diagrammes annotés
- Description technique de chaque composant :
  - PostgreSQL : schéma, génération de données
  - S3 : organisation, scripts d'export
  - Snowflake : configuration, chargement
  - dbt : modèles, tests, documentation
  - Airflow : DAG, orchestration
  - Power BI : dashboards, mesures DAX
- Extraits de code clés avec explications
- Captures d'écran des dashboards Power BI
- Réponses détaillées aux 5 questions de réflexion
- Conclusion : bilan, difficultés rencontrées, améliorations possibles
- Annexes : code complet, logs d'exécution

Livrable 2 : Dépôt Git

Format : Repository GitHub/GitLab public ou privé

Structure recommandée :

```
shopstream-data-pipeline/
├── README.md
├── docs/
│   ├── architecture.md
│   ├── setup_guide.  md
│   └── user_manual.md
├── sql/
│   ├── postgresql/
│   │   └── create_tables.sql
```

```
│   └── snowflake/
│       ├── setup. sql
│       └── load_staging.sql
├── scripts/
│   ├── generate_data.py
│   ├── export_to_s3.py
│   └── requirements.txt
├── dbt/
│   └── shopstream_dbt/
│       ├── models/
│       │   ├── staging/
│       │   ├── core/
│       │   └── marts/
│       ├── tests/
│       └── dbt_project.yml
├── airflow/
│   └── dags/
│       └── shopstream_daily_pipeline.py
├── powerbi/
│   ├── ShopStream_Dashboard.pbix
│   └── screenshots/
│       ├── dashboard_sales. png
│       ├── dashboard_products.png
│       └── dashboard_customers.png
└── . gitignore
```

Le README.md doit contenir :

- Description du projet
- Architecture (avec diagramme)
- Prérequis techniques
- Instructions d'installation étape par étape
- Instructions d'exécution
- Captures d'écran des résultats
- Technologies utilisées
- Auteurs et remerciements

Livrable 3 : Présentation (Slides + Vidéo)

Format : PowerPoint/Google Slides + Vidéo de démonstration

Slides (15-20 slides) :

- Slide 1 : Titre, auteurs, date
- Slide 2 : Contexte métier ShopStream
- Slide 3 : Problématique et objectifs
- Slides 4-6 : Architecture du pipeline (diagrammes)
- Slides 7-12 : Zoom sur chaque composant (1 slide = 1 techno)
- Slides 13-15 : Démonstration (captures d'écran des dashboards)
- Slide 16 : Comparaison on-premise vs cloud
- Slide 17 : Sécurité et RGPD
- Slide 18 : Difficultés et solutions
- Slide 19 : Améliorations futures
- Slide 20 : Conclusion et questions

Vidéo de démonstration (5-10 minutes) :

- Enregistrez votre écran avec OBS Studio ou Loom
- Montrez le pipeline en action :
  - Insertion de données dans PostgreSQL
  - Vérification dans S3
  - Vérification dans Snowflake
  - Exécution manuelle du DAG Airflow
  - Visite des dashboards Power BI
  - Interaction avec les filtres
- Ajoutez une voix off expliquant ce que vous faites
- Publiez la vidéo sur YouTube (non listée) ou Vimeo

12.4 Critères d'évaluation

Critère	Points	Description
Compréhension de l'architecture	20	Capacité à expliquer le rôle de chaque composant, les flux de données, les choix technologiques
Qualité du code	25	SQL propre et commenté, Python documenté, modèles dbt structurés, DAG Airflow fonctionnel
Dashboards BI	15	Pertinence des KPIs, lisibilité, esthétique, interactivité, utilité pour le métier
Réflexion critique	20	Qualité des réponses aux 5 questions (comparaison, KPIs, tests, RGPD), profondeur d'analyse
Documentation	10	Clarté du README, commentaires dans le code, diagrammes annotés, guide d'installation
Présentation orale	10	Clarté de l'exposé, gestion du temps, qualité de la démonstration vidéo, réponses aux questions
TOTAL	**100**	

12.5 Bonus (points supplémentaires)

- **+10 points** : Implémentation complète d'un modèle de Machine Learning intégré au pipeline (ex : prédiction du churn avec Snowflake ML ou Python)
- **+8 points** : Mise en place d'un framework de Data Quality complet (Great Expectations ou Soda Core) avec rapports automatiques

- **+5 points** : CI/CD pour dbt avec GitHub Actions (tests automatiques sur chaque PR)
- **+5 points** : Documentation dbt avancée avec descriptions, tests custom, macros réutilisables
- **+5 points** : Implémentation de Slowly Changing Dimensions (SCD Type 2) avec dbt snapshots
- **+3 points** : Monitoring avancé du pipeline avec Grafana ou DataDog
- **+3 points** : Implémentation du Row-Level Security dans Snowflake avec démonstration

12.6 Ressources complémentaires

Documentation officielle

- PostgreSQL Documentation
- Amazon S3 Documentation
- Snowflake Documentation
- dbt Documentation
- Apache Airflow Documentation
- Power BI Documentation

Livres recommandés

- **"The Data Warehouse Toolkit"** - Ralph Kimball (modélisation dimensionnelle)
- **"Designing Data-Intensive Applications"** - Martin Kleppmann (architectures distribuées)
- **"Fundamentals of Data Engineering"** - Joe Reis & Matt Housley (modern data stack)
- **"Analytics Engineering with SQL and dbt"** - Rui Machado (dbt en profondeur)
- **"Data Pipelines Pocket Reference"** - James Densmore (bonnes pratiques)

Tutoriels vidéo

- Chaîne YouTube dbt Labs
- Chaîne YouTube Snowflake
- Chaîne YouTube Apache Airflow
- Chaîne YouTube Power BI

Communautés

- dbt Community Slack (plus de 60 000 membres)
- Snowflake Community
- Stack Overflow - Apache Airflow
- Reddit r/dataengineering
- Power BI Community

> **Conseils pour réussir**
> - Commencez tôt : N'attendez pas la dernière semaine
> - Testez au fur et à mesure : Validez chaque étape avant de passer à la suivante
> - Documentez en continu : Prenez des notes et captures d'écran pendant le développement
> - Travaillez en équipe : Répartissez les tâches si c'est un projet de groupe
> - Utilisez Git : Commits réguliers, branches pour chaque feature
> - Demandez de l'aide : Forums, Slack, enseignant - n'hésitez pas
> - Privilégiez la compréhension : Mieux vaut un pipeline simple mais bien expliqué qu'un pipeline complexe non maîtrisé
> - Relisez vos livrables : Traquez les fautes, vérifiez la cohérence

12.8 FAQ et dépannage

> ### Problèmes courants et solutions

Q : Erreur de connexion PostgreSQL depuis Python

R : Vérifiez que PostgreSQL est démarré, que le mot de passe est correct, et que le port 5432 est bien ouvert.

Q : AWS CLI ne reconnaît pas mes credentials

R : Relancez `aws configure` et vérifiez que les clés sont correctement collées (pas d'espace en début/fin).

Q : Snowflake ne trouve pas mon stage S3

R : Vérifiez la Storage Integration (DESC INTEGRATION), la relation de confiance IAM, et que le bucket/région sont corrects.

Q : dbt ne trouve pas mes sources

R : Vérifiez le fichier `_staging__sources.yml`, la connexion dans `profiles.yml`, et lancez `dbt debug`.

Q : Le DAG Airflow n'apparaît pas dans l'interface

R : Vérifiez que le fichier est bien dans `dags/`, qu'il n'y a pas d'erreur de syntaxe Python, et rafraîchissez la page.

Q : Power BI ne se connecte pas à Snowflake

R : Vérifiez l'URL du serveur (sans https://), le nom du warehouse, et que l'utilisateur a les droits d'accès.

Q : Les données ne se rafraîchissent pas automatiquement

R : Vérifiez le schedule du DAG Airflow, que le scheduler est lancé, et que les connexions sont valides.

Conclusion

Ce TP vous a permis de découvrir et de manipuler un **Modern Data Stack** complet, représentatif des architectures utilisées par les entreprises innovantes aujourd'hui.

Vous avez appris à orchestrer un pipeline de données de bout en bout : de la source OLTP (PostgreSQL) jusqu'aux dashboards décisionnels (Power BI), en passant par le Data Lake (S3), le Data Warehouse (Snowflake), les transformations analytiques (dbt) et l'orchestration (Airflow).

Ces compétences sont **très recherchées sur le marché** : Analytics Engineer, Data Engineer, Solutions Architect, BI Developer. Les entreprises migrent massivement vers le cloud et ont besoin de profils maîtrisant ces technologies.

Pour aller plus loin

- **Certifications recommandées** :
  - Snowflake SnowPro Core Certification
  - AWS Certified Data Analytics - Specialty
  - dbt Analytics Engineering Certification
  - Microsoft Certified: Power BI Data Analyst Associate
  - Astronomer Certification for Apache Airflow
- **Projets personnels** :
  - Construire votre propre pipeline sur des données publiques (Kaggle, data.gouv.fr, Open Data)
  - Contribuer à des projets open-source (dbt packages, Airflow providers, Great Expectations)
  - Publier vos dashboards et analyses sur un portfolio en ligne (GitHub Pages, Notion)
  - Participer à des hackathons data
- **Communauté** :

- Participer à des meetups Data Engineering dans votre ville (Meetup. com)
- Suivre des experts sur LinkedIn et Twitter
- Lire des newsletters spécialisées (Data Engineering Weekly, dbt Roundup, Airflow Newsletter)
- Rejoindre des Slacks communautaires (dbt, Airflow, Locally Optimistic)

- **Veille technologique** :
  - Suivre les annonces des conférences (Snowflake Summit, AWS re:Invent, Coalesce by dbt)
  - Lire des blogs techniques (dbt Labs, Airbyte, Fivetran, Hightouch)
  - Expérimenter avec de nouveaux outils (Dagster, Prefect, Mage.ai)

Bon courage pour la réalisation de ce TP N'hésitez pas à poser des questions et à expérimenter