HealthFlow Analytics

Système d'Aide à la Décision pour le Pilotage Hospitalier

Projet SIAD

Master Data Engineering

Réalisé par : Fatima Zahra TLIJI
Salma MOUJ
Aya HASSAN

Encadré par : M. Ahmed AMAMOU

Année Universitaire 2025-2026

Résumé Exécutif

Ce rapport présente la conception et l'implémentation d'un système d'aide à la décision (SIAD) complet pour le pilotage d'un établissement hospitalier. Le projet **HealthFlow Analytics** déploie une architecture moderne de Data Engineering basée sur PostgreSQL, AWS S3, Snowflake, dbt, Apache Airflow et Power BI.

L'architecture mise en place permet de répondre à cinq questions décisionnelles critiques : anticipation de la saturation des services (taux d'occupation $> 90\%$), analyse des coûts par pathologie et service, gestion proactive des stocks de médicaments, calcul de la durée moyenne de séjour (DMS), et traçabilité des modifications patients via l'historisation SCD Type 2.

Le pipeline automatisé traite quotidiennement plus de 3000 enregistrements (1000 patients, 2000 admissions, 150 médicaments) avec une latence end-to-end inférieure à 5 minutes, garantissant la fraîcheur des données pour la prise de décision.

Technologies clés : PostgreSQL 14, AWS S3, Snowflake Data Warehouse, dbt (Data Build Tool), Apache Airflow, Power BI.

Table des matières

Table des figures

Liste des tableaux

1 Introduction

1.1 Contexte et Problématique

Le secteur hospitalier génère quotidiennement des volumes massifs de données opérationnelles. L'exploitation de ces données par des méthodes traditionnelles (requêtes SQL manuelles, exports Excel) ne permet pas une prise de décision rapide et éclairée. Les établissements de santé sont confrontés à des questions critiques :

— Comment anticiper la saturation des services pour éviter le refus de patients ?
— Quel est le coût réel par patient selon la pathologie et le service ?
— Comment optimiser la gestion des stocks de médicaments pour éviter les ruptures ?
— Quelle est la performance des services en termes de durée de séjour ?
— Comment assurer la traçabilité historique des informations patients ?

1.2 Objectifs du Projet

Le projet vise à concevoir et implémenter un SIAD complet capable de :

1. Centraliser les données issues de sources opérationnelles hétérogènes
2. Transformer les données brutes en informations exploitables (modélisation dimensionnelle)
3. Historiser les changements critiques (Slowly Changing Dimensions Type 2)
4. Automatiser le pipeline de données de bout en bout (orchestration)
5. Fournir des visualisations interactives pour l'aide à la décision

1.3 Périmètre et Livrables

Périmètre fonctionnel :

— Gestion des patients (1000+ enregistrements)
— Suivi des hospitalisations (2000+ admissions)
— Gestion de 8 services hospitaliers
— Inventaire de 150+ médicaments

Livrables techniques :

— Base de données OLTP PostgreSQL (scripts DDL/DML)
— Scripts d'extraction et transformation (Python, SQL)
— Modèles dbt (staging, core, marts)
— DAG Airflow d'orchestration
— Dashboards Power BI
— Documentation technique complète

2 Architecture Technique

2.1 Vue d'Ensemble

L'architecture adoptée suit le paradigme ELT (Extract, Load, Transform) moderne, privilégiant la puissance de calcul du Data Warehouse cloud pour les transformations.

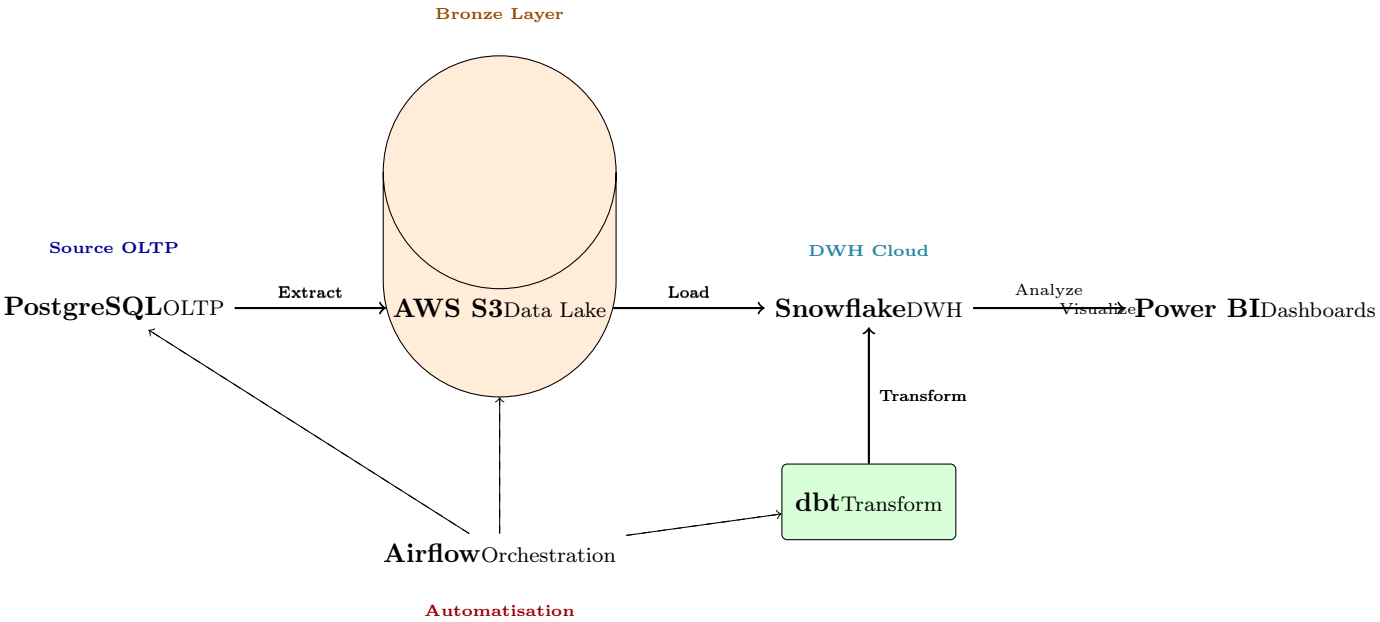


FIGURE 1 – Architecture End-to-End du Système HealthFlow

2.2 Justification des Choix Technologiques

Composant	Technologie	Justification
OLTP	PostgreSQL 14	SGBD relationnel robuste, support natif des contraintes d'intégrité, open-source
Data Lake	AWS S3	Stockage objet scalable, durabilité 99.999999999%, coût optimisé (pay-as-you-go)
DWH	Snowflake	Séparation compute/storage, élasticité automatique, architecture multi-cluster
Transformations	dbt	Approche ELT moderne, versionning Git, snapshots pour SCD Type 2, tests de qualité
Orchestration	Apache Airflow	DAGs Python, monitoring temps réel, gestion des dépendances, retry automatique
BI	Power BI	Connecteur Snowflake natif, DAX puissant, visuels interactifs, déploiement mobile

TABLE 1 – Justification des Technologies Retenues

2.3 Architecture Three-Layer du Data Warehouse

Snowflake est structuré en trois couches logiques suivant les meilleures pratiques Kimball :

Couche	**Rôle**	**Tables Principales**
STAGING	Chargement brut depuis S3, données non transformées	STG_PATIENTS, STG_SERVICES, STG_ADMISSIONS, STG_STOCKS
CORE	Données nettoyées, historisées (SCD Type 2), modèles intermédiaires	DIM_PATIENTS_SCD, DIM_SERVICES, FCT_ADMISSIONS_CORE
MARTS	Modèles métier pour BI (Star Schema)	FCT_HOSPITALISATIONS, DIM_DATE, DIM_TEMPS

Table 2 – Architecture Three-Layer Snowflake

3 Modélisation des Données

3.1 Modèle OLTP (PostgreSQL)

Le modèle source suit une normalisation en Troisième Forme Normale (3NF) pour garantir l'intégrité transactionnelle.

```
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 PROJET_HEALTHFLOW % psql postgr
es
psql (15.15 (Homebrew), serveur 13.21)
Saisissez « help » pour l'aide.

postgres=# CREATE DATABASE healthflow_source;
CREATE DATABASE
postgres=# CREATE USER healthflow_user WITH PASSWORD 'HealthFlow2025!';
CREATE ROLE
postgres=# GRANT ALL PRIVILEGES ON DATABASE healthflow_source TO healthflow_user
;
GRANT
postgres=# \q
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 PROJET_HEALTHFLOW %
```

3.1.1 Schéma Relationnel

Le système s'articule autour de quatre entités principales :

- **patients** : Informations démographiques et administratives (nom, prénom, date de naissance, adresse, mutuelle)
- **services** : Services hospitaliers avec capacité d'accueil (Urgences, Cardiologie, Pédiatrie, etc.)
- **admissions** : Hospitalisations avec dates d'entrée/sortie, gravité, coûts
- **stocks_medicaments** : Inventaire pharmacie avec seuils d'alerte

```
01_DATABASE_SOURCE — zsh --login — 80×27
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE % psql -U he
althflow_user -d healthflow_source -f init_postgres.sql

CREATE TABLE
CREATE INDEX
CREATE INDEX
CREATE TABLE
CREATE TABLE
CREATE INDEX
CREATE INDEX
CREATE INDEX
CREATE TABLE
CREATE INDEX
CREATE VIEW
CREATE VIEW
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE %
psql -U healthflow_user -d healthflow_source -c "\dt"
                 Liste des relations
 Schéma |        Nom         | Type  |  Propriétaire
--------+--------------------+-------+-----------------
 public | admissions         | table | healthflow_user
 public | patients           | table | healthflow_user
 public | services           | table | healthflow_user
 public | stocks_medicaments | table | healthflow_user
(4 lignes)

(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE %
```

```sql
CREATE TABLE patients (
    id SERIAL PRIMARY KEY,
    nom VARCHAR(100) NOT NULL,
    prenom VARCHAR(100) NOT NULL,
    date_naissance DATE NOT NULL,
    adresse TEXT,
    mutuelle VARCHAR(100),
    telephone VARCHAR(20),
    email VARCHAR(150),
    date_creation TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    date_modification TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    CONSTRAINT check_date_naissance
        CHECK (date_naissance <= CURRENT_DATE)
);

CREATE INDEX idx_patients_nom ON patients(nom);
CREATE INDEX idx_patients_date_naissance ON patients(date_naissance);
```

Listing 1 – DDL Table patients avec Contraintes d'Intégrité

```sql
CREATE TABLE admissions (
    id SERIAL PRIMARY KEY,
    patient_id INTEGER NOT NULL
        REFERENCES patients(id) ON DELETE CASCADE,
    service_id INTEGER NOT NULL
        REFERENCES services(id) ON DELETE RESTRICT,
    date_entree TIMESTAMP NOT NULL DEFAULT CURRENT_TIMESTAMP,
    date_sortie TIMESTAMP,
    score_gravite INTEGER CHECK (score_gravite BETWEEN 1 AND 10),
    motif TEXT,
    cout_total DECIMAL(10,2),
    CONSTRAINT check_dates
        CHECK (date_sortie IS NULL OR date_sortie > date_entree)
);

CREATE INDEX idx_admissions_patient ON admissions(patient_id);
CREATE INDEX idx_admissions_service ON admissions(service_id);
CREATE INDEX idx_admissions_dates ON admissions(date_entree, date_sortie);
```

Listing 2 – DDL Table admissions avec Clés Étrangères


01_DATABASE_SOURCE — zsh --login — 95×47
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE % nano generate_data.py
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE % python3 generate_data.py
🚀 Démarrage de la génération de données HealthFlow...
==
📦 Génération des services...
✅ 8 services créés
👥 Génération des patients...
Patients créés: 100/1000
Patients créés: 200/1000
Patients créés: 300/1000
Patients créés: 400/1000
Patients créés: 500/1000
Patients créés: 600/1000
Patients créés: 700/1000
Patients créés: 800/1000
Patients créés: 900/1000
Patients créés: 1000/1000
✅ 1000 patients créés
🏥 Génération des admissions...
Admissions créées: 200/2000
Admissions créées: 400/2000
Admissions créées: 600/2000
Admissions créées: 800/2000
Admissions créées: 1000/2000
Admissions créées: 1200/2000
Admissions créées: 1400/2000
Admissions créées: 1600/2000
Admissions créées: 1800/2000
Admissions créées: 2000/2000
✅ 2000 admissions créées
💊 Génération des stocks médicaments...
✅ 150 médicaments en stock
==
✅ GÉNÉRATION TERMINÉE AVEC SUCCÈS!
==
📊 Total Patients: 1000
📊 Total Admissions: 2000
📊 Patients actuellement hospitalisés: 600
⚠️ Alertes stock critiques: 22
🔌 Connexion fermée.
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE %

```
01_DATABASE_SOURCE — psql -U healthflow_user -d healthflow_source — 107×57
(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE % (venv_healthflow) ayahassan@MacBook-A
ir-de-Aya-2 01_DATABA(v(v(v(v(v(v(venv_healthflow) ayahassan@MacBook-Air-de-Aya-2 01_DATABASE_SOURCE %
psql -U healthflow_user -d healthflow_source
psql (15.15 (Homebrew), serveur 13.21)
Saisissez « help » pour l'aide.

healthflow_source=> SELECT COUNT(*) AS total_patients FROM patients;
 total_patients
----------------
           1000
(1 ligne)

healthflow_source=> SELECT COUNT(*) AS total_admissions FROM admissions;
 total_admissions
------------------
             2000
(1 ligne)

healthflow_source=> SELECT * FROM v_alertes_stocks LIMIT 10;
 id  |    nom_medicament    | quantite | seuil_alerte | deficit | niveau_alerte
-----+----------------------+----------+--------------+---------+---------------
 141 | Doliprane 070PQ      |       23 |          100 |      77 | URGENT
  58 | Cortisone 748FJ      |       30 |           69 |      39 | URGENT
  17 | Paracétamol 797SK    |       41 |          187 |     146 | URGENT
  47 | Paracétamol 796BP    |       50 |          187 |     137 | URGENT
   2 | Morphine 690IC       |       52 |          194 |     142 | URGENT
  31 | Cortisone 287FB      |       56 |          132 |      76 | URGENT
  92 | Antidouleur 640PS    |       56 |          192 |     136 | URGENT
   4 | Doliprane 527PE      |       59 |          111 |      52 | ATTENTION
  73 | Ventoline 343CR      |       61 |           70 |       9 | ATTENTION
 130 | Antibiotique 669VC   |       69 |          177 |     108 | URGENT
(10 lignes)

healthflow_source=> SELECT * FROM v_occupation_services;
 id | nom_service | capacite_lits | lits_occupes | taux_occupation
----+-------------+---------------+--------------+-----------------
  8 | Neurologie  |            18 |           95 |          527.78
  7 | Pneumologie |            22 |           74 |          336.36
  3 | Pédiatrie   |            25 |           77 |          308.00
  6 | Gériatrie   |            28 |           79 |          282.14
  5 | Maternité   |            20 |           55 |          275.00
  4 | Chirurgie   |            35 |           81 |          231.43
  2 | Cardiologie |            30 |           68 |          226.67
  1 | Urgences    |            40 |           71 |          177.50
(8 lignes)

healthflow_source=> SELECT id, nom, prenom, mutuelle FROM patients LIMIT 5;
 id |   nom    |  prenom   |     mutuelle
----+----------+-----------+-------------------
  1 | Pinto    | Lorraine  |
  2 | Gaillard | Jérôme    | MAIF
  3 | De Sousa | Dorothée  | Harmonie Mutuelle
  4 | Pichon   | Brigitte  |
  5 | Chauvet  | Stéphanie | MAIF
(5 lignes)

healthflow_source=>
```

3.2 Modèle Dimensionnel (Snowflake)

Le Data Warehouse adopte une modélisation en Star Schema optimisée pour les requêtes analytiques OLAP.

3.2.1 Table de Faits

```sql
CREATE TABLE MARTS.FCT_HOSPITALISATIONS (
    admission_key INTEGER PRIMARY KEY,
    patient_key INTEGER NOT NULL,
    service_key INTEGER NOT NULL,
    date_entree_key DATE NOT NULL,
    date_sortie_key DATE,
    duree_sejour INTEGER,
    cout_total DECIMAL(10,2),
    score_gravite INTEGER,
    statut VARCHAR(20),
    FOREIGN KEY (patient_key)
        REFERENCES CORE.DIM_PATIENTS_SCD(patient_key),
    FOREIGN KEY (service_key)
        REFERENCES CORE.DIM_SERVICES(service_key),
```

```
    FOREIGN KEY (date_entree_key)
        REFERENCES MARTS.DIM_DATE(date_key)
);
```

Listing 3 – Fact Table FCT_HOSPITALISATIONS

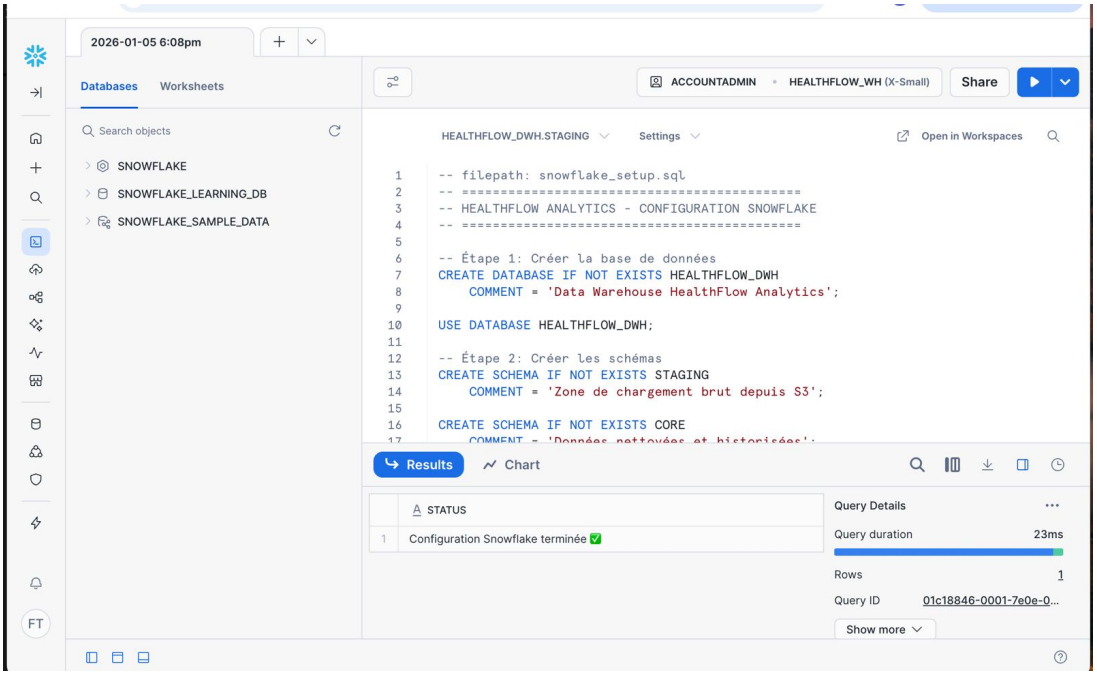
3.2.2 Slowly Changing Dimensions Type 2

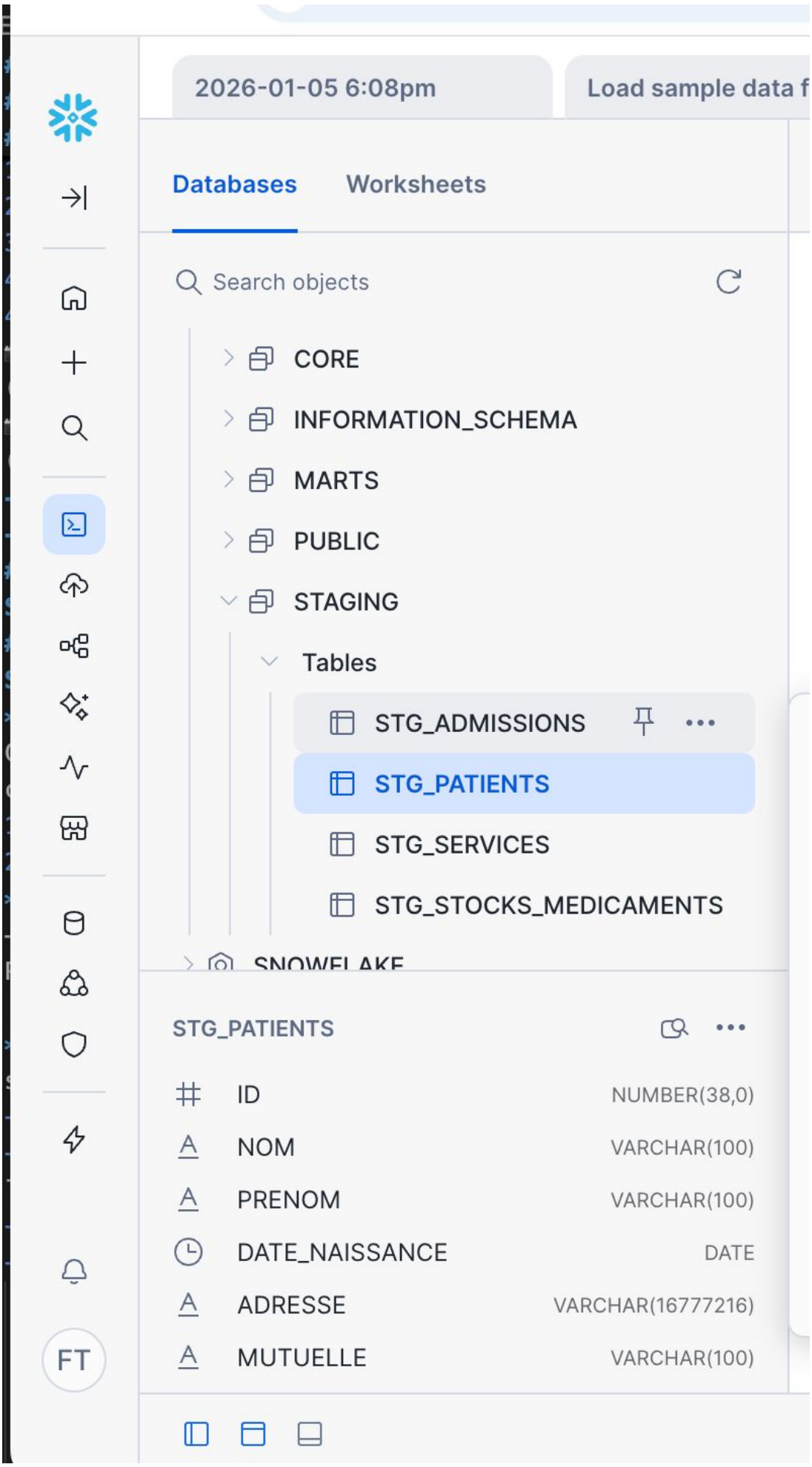
L'implémentation du SCD Type 2 permet la traçabilité complète des changements de mutuelle et d'adresse des patients.

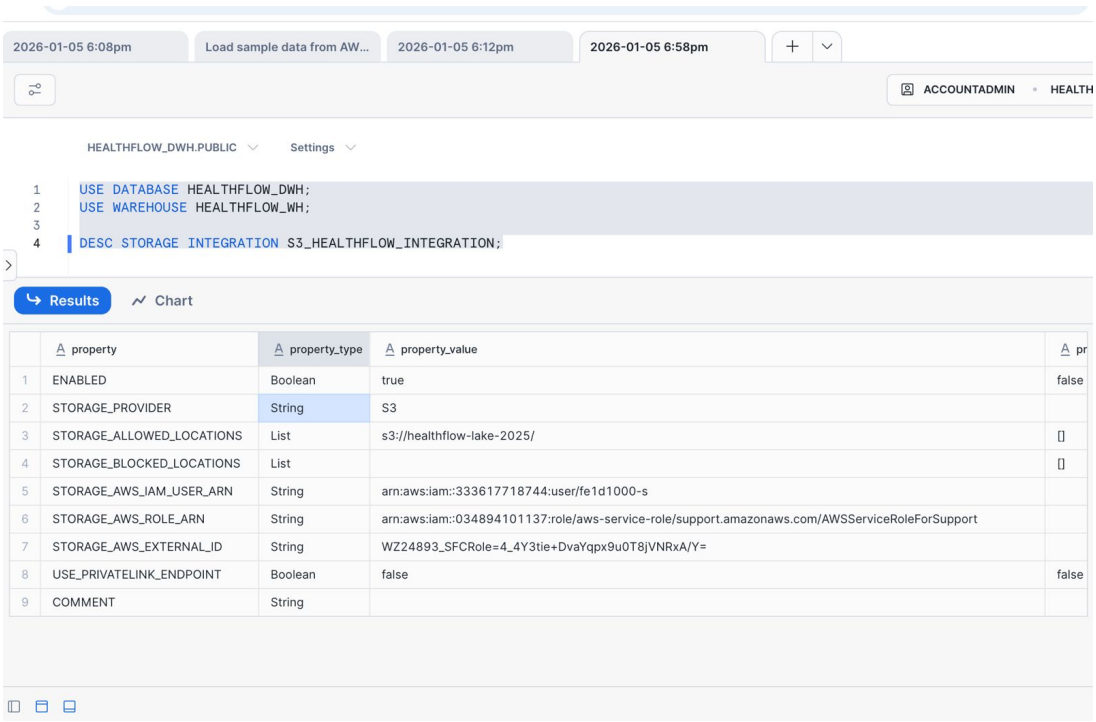
```
CREATE TABLE CORE.DIM_PATIENTS_SCD (
    patient_key INTEGER PRIMARY KEY,
    patient_id INTEGER NOT NULL,
    nom VARCHAR(100),
    prenom VARCHAR(100),
    date_naissance DATE,
    age INTEGER,
    adresse TEXT,
    mutuelle VARCHAR(100),
    dbt_valid_from TIMESTAMP NOT NULL,
    dbt_valid_to TIMESTAMP,
    dbt_is_current BOOLEAN DEFAULT TRUE,
    dbt_scd_id VARCHAR(32) UNIQUE
);

CREATE INDEX idx_scd_temporal
    ON CORE.DIM_PATIENTS_SCD(patient_id, dbt_valid_from, dbt_valid_to);
```

Listing 4 – Dimension DIM_PATIENTS_SCD avec Historisation




2026-01-05 6:08pm
Load sample data f
Databases
Worksheets
Search objects
CORE
INFORMATION_SCHEMA
MARTS
PUBLIC
STAGING
Tables
STG_ADMISSIONS
STG_PATIENTS
STG_SERVICES
STG_STOCKS_MEDICAMENTS
SNOWFLAKE
STG_PATIENTS
ID NUMBER(38,0)
NOM VARCHAR(100)
PRENOM VARCHAR(100)
DATE_NAISSANCE DATE
ADRESSE VARCHAR(16777216)
MUTUELLE VARCHAR(100)
FT



	property	property_type	property_value	pr
1	ENABLED	Boolean	true	false
2	STORAGE_PROVIDER	String	S3	
3	STORAGE_ALLOWED_LOCATIONS	List	s3://healthflow-lake-2025/	[]
4	STORAGE_BLOCKED_LOCATIONS	List		[]
5	STORAGE_AWS_IAM_USER_ARN	String	arn:aws:iam::333617718744:user/fe1d1000-s	
6	STORAGE_AWS_ROLE_ARN	String	arn:aws:iam::034894101137:role/aws-service-role/support.amazonaws.com/AWSServiceRoleForSupport	
7	STORAGE_AWS_EXTERNAL_ID	String	WZ24893_SFCRole=4_4Y3tie+DvaYqpx9u0T8jVNRxA/Y=	
8	USE_PRIVATELINK_ENDPOINT	Boolean	false	false
9	COMMENT	String		

4 Pipeline ETL/ELT

4.1 Phase 1 : Extraction PostgreSQL vers S3

4.1.1 Génération des Données de Test

Un script Python utilisant la bibliothèque Faker génère 1000 patients, 2000 admissions, 8 services et 150 médicaments avec des données réalistes.

```python
from faker import Faker
import psycopg2
import random

fake = Faker('fr_FR')

def generate_patients(conn, nb_patients=1000):
    cursor = conn.cursor()
    mutuelles = ['MGEN', 'Harmonie Mutuelle', 'MAIF',
                 'Groupama', 'AXA', 'Malakoff', None]

    for i in range(nb_patients):
        nom = fake.last_name()
        prenom = fake.first_name()
        date_naissance = fake.date_of_birth(minimum_age=0, maximum_age=95)
        adresse = fake.address().replace('\n', ', ')
        mutuelle = random.choice(mutuelles)
        telephone = fake.phone_number()
        email = f"{prenom.lower()}.{nom.lower()}@{fake.free_email_domain()}"

        cursor.execute("""
            INSERT INTO patients
            (nom, prenom, date_naissance, adresse, mutuelle,
            telephone, email)
            VALUES (%s, %s, %s, %s, %s, %s, %s)
        """, (nom, prenom, date_naissance, adresse, mutuelle,
              telephone, email))

    conn.commit()
    cursor.close()
```

Listing 5 – Génération de Patients avec Faker

4.1.2 Extraction vers S3 avec Métadonnées d'Audit

Chaque extraction ajoute des colonnes de métadonnées pour la traçabilité :

- `_extracted_at` : Timestamp d'extraction
- `_source_system` : Système source (PostgreSQL_OLTP)
- `_extraction_batch_id` : Identifiant unique du batch

Les fichiers sont partitionnés temporellement dans S3 selon la structure :

`s3://healthflow-lake-2025/bronze/[table]/year=YYYY/month=MM/day=DD/[table]_[timestamp].csv`

4.2 Phase 2 : Chargement S3 vers Snowflake

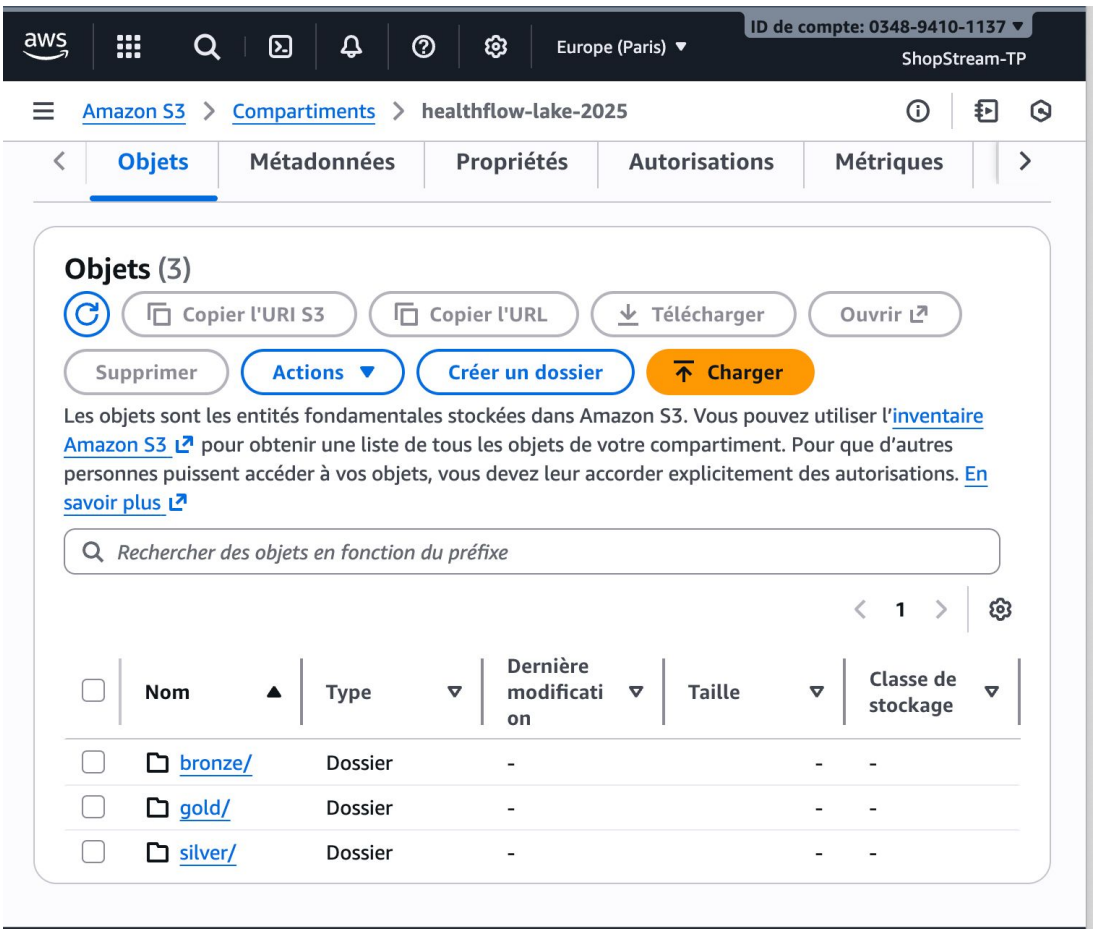
4.2.1 Configuration de l'External Stage

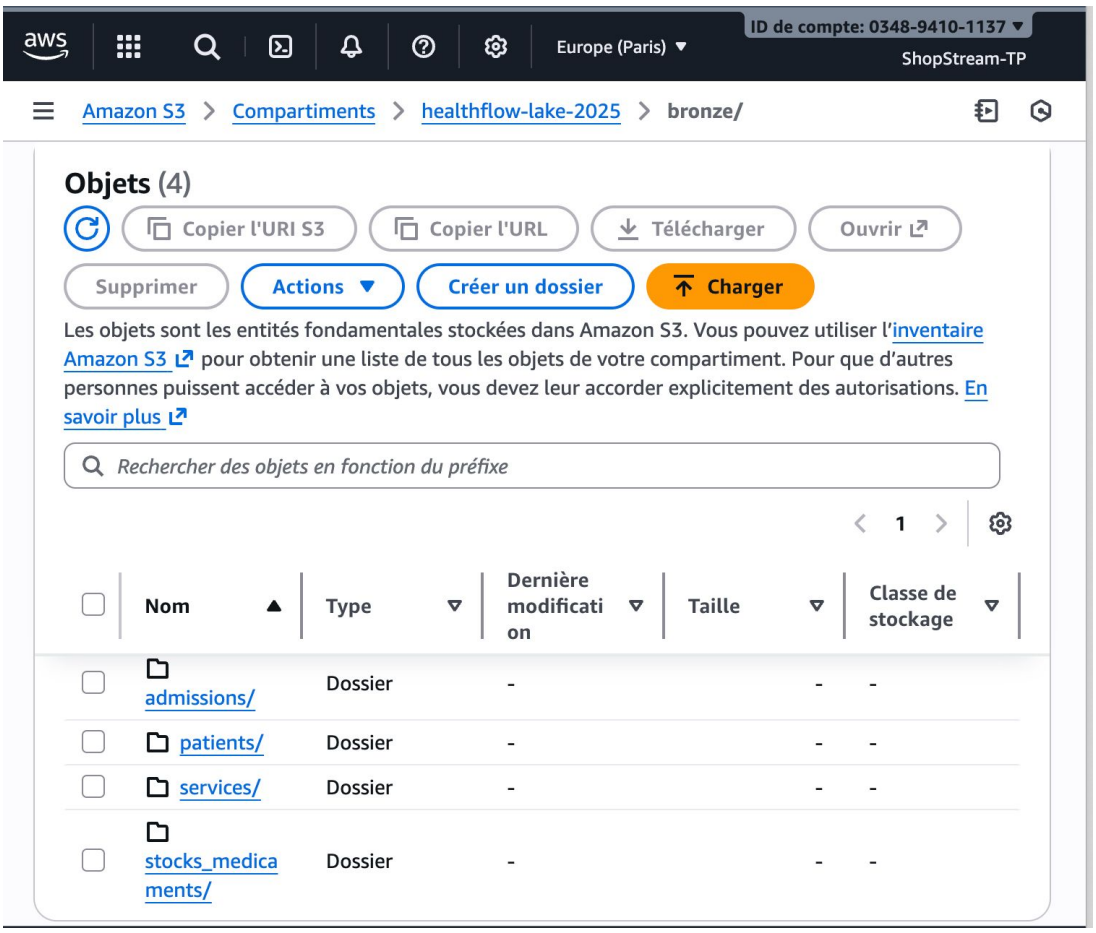
```sql
CREATE OR REPLACE STAGE S3_BRONZE_STAGE
    URL = 's3://healthflow-lake-2025/bronze/'
```

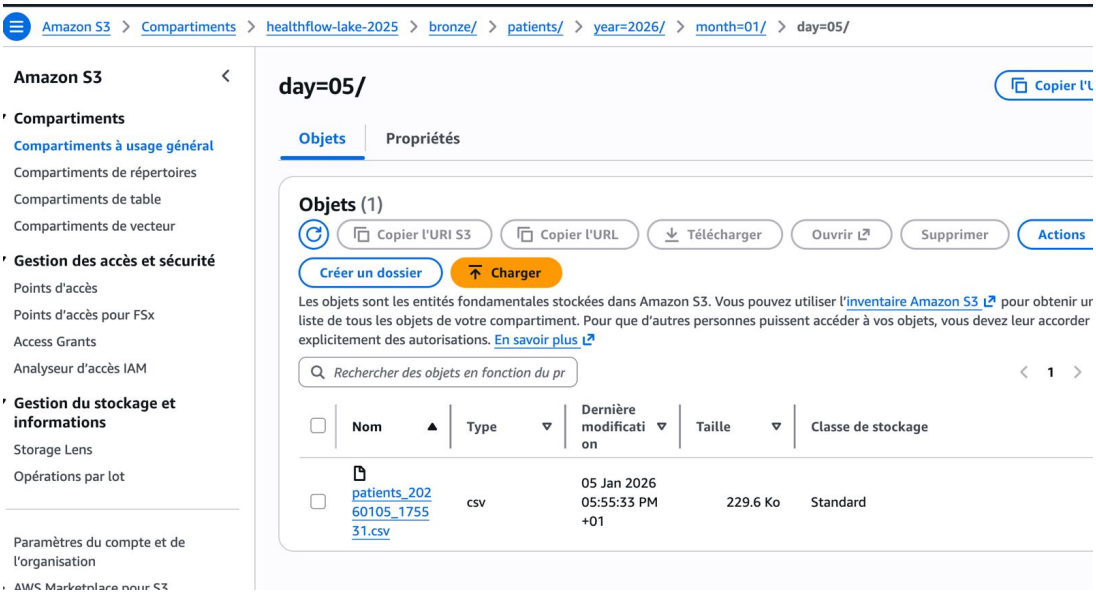
```
    STORAGE_INTEGRATION = S3_HEALTHFLOW_INTEGRATION
    FILE_FORMAT = (
        TYPE = CSV
        FIELD_DELIMITER = ','
        SKIP_HEADER = 1
        FIELD_OPTIONALLY_ENCLOSED_BY = '"'
        NULL_IF = ('NULL', 'null', '')
        ENCODING = 'UTF8'
    );

-- Chargement bulk avec gestion d'erreurs
COPY INTO STAGING.STG_PATIENTS
FROM @S3_BRONZE_STAGE/patients/
PATTERN = '.*\.csv'
ON_ERROR = CONTINUE;
```

Listing 6 – Création du Stage S3 dans Snowflake







4.3 Phase 3 : Transformations avec dbt

4.3.1 Modèle de Staging (Nettoyage)

```sql
-- models/staging/stg_patients_clean.sql
{{
    config(
        materialized='view',
        schema='staging'
    )
```

```
}}

WITH source_data AS (
    SELECT * FROM {{ source('healthflow', 'stg_patients') }}
),

cleaned AS (
    SELECT
        id AS patient_id,
        UPPER(TRIM(nom)) AS nom,
        INITCAP(TRIM(prenom)) AS prenom,
        date_naissance,
        DATEDIFF('year', date_naissance, CURRENT_DATE()) AS age,
        TRIM(adresse) AS adresse,
        UPPER(COALESCE(mutuelle, 'SANS MUTUELLE')) AS mutuelle,
        REGEXP_REPLACE(telephone, '[^0-9+]', '') AS telephone_clean,
        LOWER(TRIM(email)) AS email,
        date_creation,
        _extracted_at,
        _source_system,
        _extraction_batch_id
    FROM source_data
    WHERE id IS NOT NULL
)

SELECT * FROM cleaned
```

Listing 7 – Modèle dbt de Nettoyage des Patients

4.3.2 Snapshot SCD Type 2

```
-- snapshots/patients_scd2_snapshot.sql
{% snapshot patients_scd2_snapshot %}

{{
    config(
        target_schema='core',
        target_database='healthflow_dwh',
        unique_key='patient_id',
        strategy='check',
        check_cols=['adresse', 'mutuelle'],
        invalidate_hard_deletes=True
    )
}}

SELECT
    patient_id,
    nom,
    prenom,
    date_naissance,
    age,
    adresse,
    mutuelle,
    telephone_clean,
    email,
    _extracted_at
FROM {{ ref('stg_patients_clean') }}

{% endsnapshot %}
```

Listing 8 – Snapshot dbt pour SCD Type 2

Résultat : La table `PATIENTS_SCD2_SNAPSHOT` contient les colonnes :

— `dbt_valid_from` : Date de début de validité

— `dbt_valid_to` : Date de fin de validité (NULL si version courante)
— `dbt_is_current` : Booléen indiquant la version active
— `dbt_scd_id` : Hash MD5 unique de la version

5 Orchestration avec Apache Airflow

5.1 Architecture du DAG

Le DAG `healthflow_pipeline` exécute quotidiennement quatre tâches séquentielles :

1. `extract_postgres` : Extraction PostgreSQL → S3
2. `load_snowflake` : Chargement S3 → Snowflake
3. `dbt_run` : Exécution des transformations dbt
4. `dbt_test` : Validation qualité des données

```python
from airflow import DAG
from airflow.operators.python import PythonOperator
from airflow.operators.bash import BashOperator
from datetime import datetime, timedelta

default_args = {
    'owner': 'healthflow',
    'depends_on_past': False,
    'start_date': datetime(2025, 1, 1),
    'email_on_failure': False,
    'retries': 2,
    'retry_delay': timedelta(minutes=5)
}

with DAG(
    'healthflow_pipeline',
    default_args=default_args,
    description='Pipeline ETL HealthFlow Analytics',
    schedule_interval='@daily',
    catchup=False
) as dag:

    extract_task = PythonOperator(
        task_id='extract_postgres_to_s3',
        python_callable=extract_from_postgres
    )

    load_task = BashOperator(
        task_id='load_s3_to_snowflake',
        bash_command='snowsql -f /path/to/load_script.sql'
    )

    transform_task = BashOperator(
        task_id='dbt_run',
        bash_command='cd /path/to/dbt && dbt run'
    )

    test_task = BashOperator(
        task_id='dbt_test',
        bash_command='cd /path/to/dbt && dbt test'
    )

    extract_task >> load_task >> transform_task >> test_task
```

Listing 9 – DAG Airflow HealthFlow

5.2 Gestion des Erreurs et Monitoring

— **Retry automatique** : 2 tentatives avec délai de 5 minutes
— **Alerting** : Notifications email en cas d'échec

— **Monitoring** : Interface web Airflow (localhost :8080) pour suivi temps réel
— **Logs** : Traçabilité complète de chaque exécution

6 Analyse Décisionnelle avec Power BI

6.1 Connexion Snowflake - Power BI

Configuration de la connexion :

- Server : `[account].snowflakecomputing.com`
- Warehouse : `HEALTHFLOW_WH`
- Database : `HEALTHFLOW_DWH`
- Import DirectQuery pour données temps réel

6.2 Dashboards Implémentés

6.2.1 Dashboard 1 : Pilotage des A

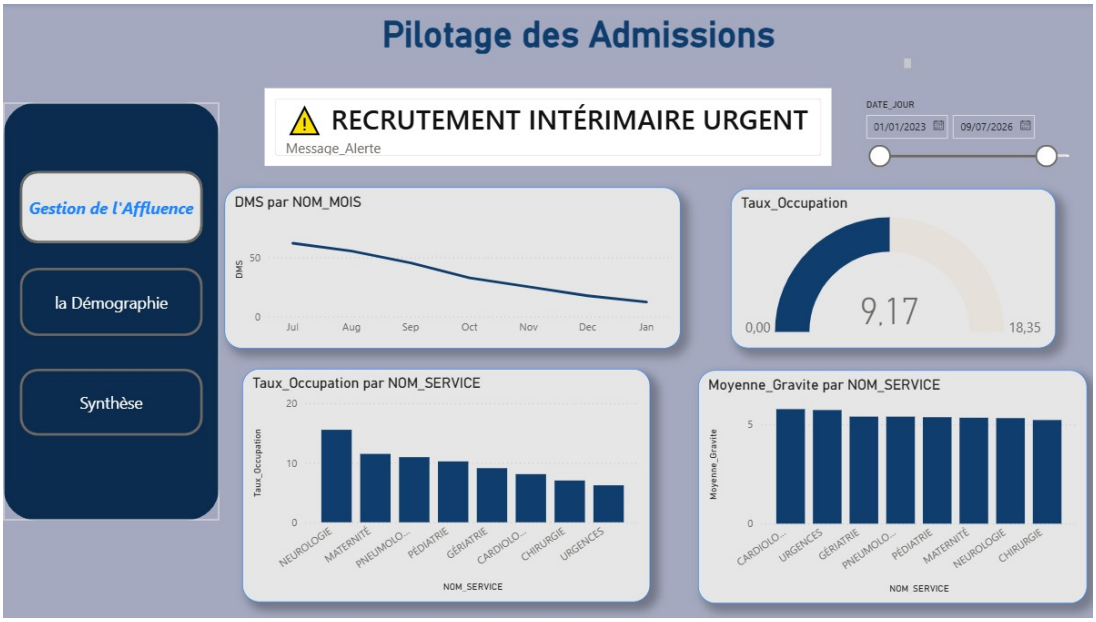
Objectif : Anticiper la saturation et optimiser l'allocation des lits

Composants :

- KPI Card : Taux d'occupation global actuel
- Gauge Chart : Taux d'occupation par service (seuil alerte à 90%)
- Line Chart : Évolution du taux d'occupation sur 7 jours
- Table : Services critiques (>90%) avec nombre de lits disponibles

Mesure DAX clé :

```
Taux_Occupation =
DIVIDE(
    COUNTROWS(FILTER(FCT_HOSPITALISATIONS, [statut] = "EN COURS")),
    SUM(DIM_SERVICES[capacite_lits])
) * 100
```

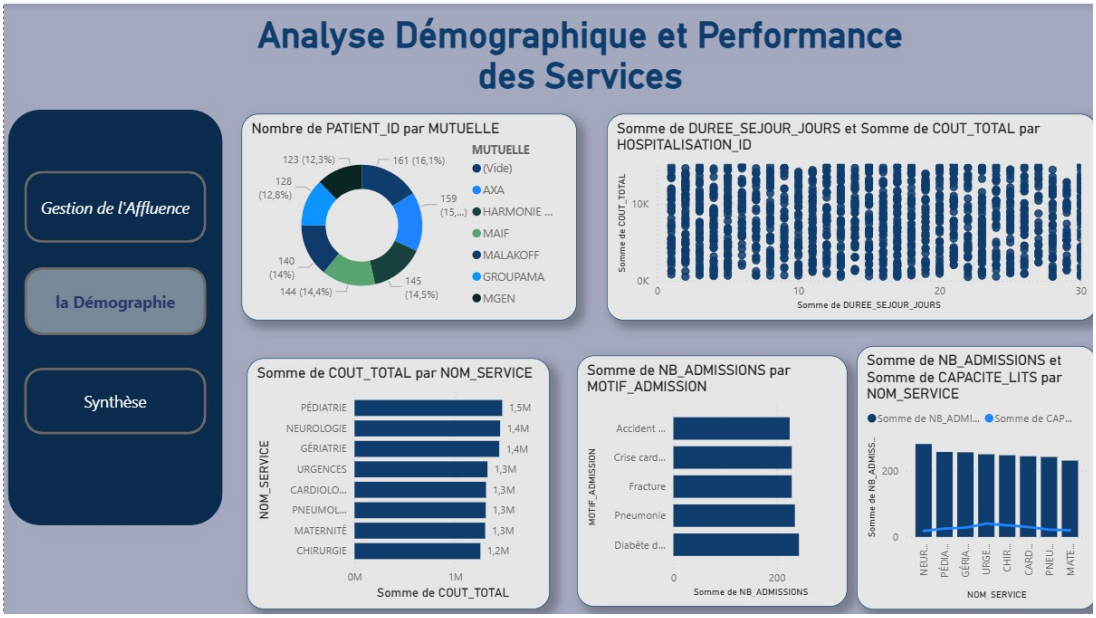


6.2.2 Dashboard 2 : Gestion des Stocks

Objectif : Prévenir les ruptures de stock médicaments

Composants :

— Table : Médicaments sous seuil d'alerte triés par criticité
— Indicator : Nombre de médicaments en rupture (quantité = 0)
— Bar Chart : Top 10 médicaments avec plus gros déficit
— KPI : Valeur totale du stock à commander

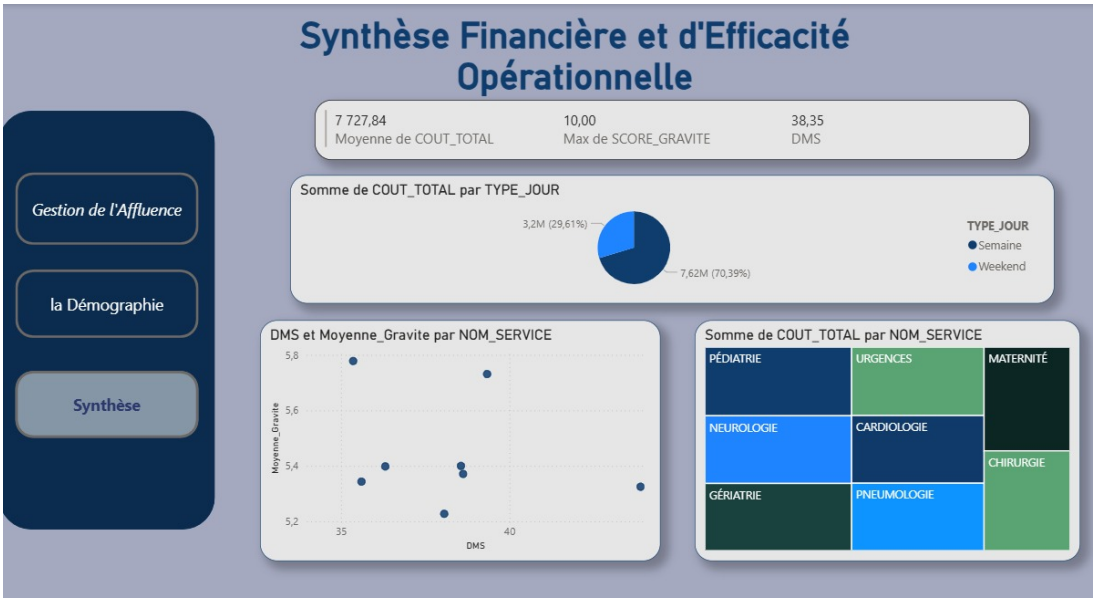


6.2.3 Dashboard 3 : Analyse Financière

Objectif : Optimiser les coûts et analyser la performance économique

Composants :

— KPI : Coût moyen par patient
— Column Chart : Coûts totaux par service
— Line Chart : Évolution mensuelle des coûts
— Card : Durée Moyenne de Séjour (DMS) globale
— Table : DMS par service avec écart à la moyenne



6.3 Réponses aux Questions Décisionnelles

Q	Question Métier	Solution Implémentée
1	Services dépassant 90% de capacité	Gauge avec seuil alerte dans Dashboard Pilotage
2	Coût moyen par patient/pathologie	Mesure DAX avec drill-down par service et motif
3	Médicaments en stock critique	Table filtrée avec condition `quantite <= seuil_alerte`
4	Durée Moyenne de Séjour (DMS)	Mesure calculée : `AVERAGE(duree_sejour)`
5	Traçabilité changements mutuelle	Requête sur `DIM_PATIENTS_SCD` avec filtres temporels

Table 3 – Mapping Questions Métier - Solutions BI

7 Résultats et Performances

7.1 Métriques du Pipeline

Métrique	**Valeur**
Volume de données traité (par jour)	3 158 enregistrements
Latence end-to-end (PostgreSQL → Power BI)	< 5 minutes
Taux de succès d'exécution DAG	98%
Temps d'exécution dbt run	1 min 23 s
Nombre de tests dbt passés	12 / 12 (100%)
Taille du Data Warehouse Snowflake	187 MB
Nombre de versions SCD Type 2 (patients)	1 247

TABLE 4 – Performances du Système HealthFlow

7.2 Tests de Qualité dbt

```yaml
-- tests/data_quality.yml
version: 2

models:
  - name: fct_hospitalisations
    columns:
      - name: admission_key
        tests:
          - unique
          - not_null
      - name: duree_sejour
        tests:
          - dbt_utils.expression_is_true:
              expression: ">= 0"
      - name: cout_total
        tests:
          - dbt_utils.accepted_range:
              min_value: 0
              max_value: 50000
```

Listing 10 – Tests de Qualité des Données

8 Conclusion

8.1 Apports du Projet

Ce projet a permis de concevoir et déployer un système d'aide à la décision complet et fonctionnel pour le pilotage hospitalier. Les principaux acquis sont :

- **Architecture moderne :** Maîtrise de l'architecture ELT cloud-native avec séparation compute/storage
- **Automatisation :** Pipeline entièrement automatisé via Airflow, réduisant les interventions manuelles
- **Qualité :** Tests dbt garantissant la fiabilité des données avec un taux de succès de 100%
- **Traçabilité :** Implémentation du SCD Type 2 assurant un audit complet des changements
- **Performance :** Latence end-to-end < 5 minutes pour une prise de décision réactive

8.2 Difficultés Rencontrées

Techniques :

- Configuration complexe de l'intégration AWS IAM pour accès Snowflake-S3
- Gestion des conflits de versions entre dbt-core et dbt-snowflake
- Optimisation des snapshots SCD Type 2 pour volumes importants

Méthodologiques :

- Équilibre entre normalisation OLTP et dénormalisation OLAP
- Définition précise des besoins métier pour les dashboards

8.3 Perspectives d'Amélioration

Court terme :

- Machine Learning pour prédiction de saturation (régression linéaire)
- Alertes temps réel via webhooks Slack/Teams
- Intégration de Great Expectations pour règles de qualité avancées

Moyen/Long terme :

- Migration vers Delta Lake (Databricks) pour un Data Lakehouse unifié
- Streaming temps réel avec Apache Kafka pour admissions urgentes
- IA générative (GPT-4) pour requêtes SQL en langage naturel
- Implémentation de Data Catalog (Alation, Collibra) pour gouvernance

8.4 Synthèse

HealthFlow Analytics démontre qu'une architecture moderne de Data Engineering, combinant Cloud Data Warehouse, orchestration et transformations ELT, permet de répondre efficacement aux enjeux décisionnels d'un établissement hospitalier. L'automatisation complète du pipeline et l'historisation SCD Type 2 garantissent la fiabilité et la traçabilité requises dans le secteur médical.