Rapport Professionnel E6

Evolution d'un Entrepot de Donnees

Projet Data Engineering
Analyse Territoriale – Region Hauts-de-France

Competences evaluees :

C16 Gerer l'entrepot de donnees a l'aide des outils d'administration et de supervision dans le respect du RGPD, afin de garantir les bons acces, l'integration des evolutions structurelles et son maintien en condition operationnelle dans le temps.

C17 Implementer des variations dans les dimensions de l'entrepot de donnees en appliquant la methode adaptee en fonction du type de changement demande afin d'historiser les evolutions de l'activite de l'organisation et maintenir ainsi une bonne capacite d'analyse.

Auteur : Hamza Elbrek
Formation : Data Engineer
Date : Fevrier 2026

Table des matières

1 Introduction

Le projet E5 avait jete les bases d'un entrepot de donnees (Data Warehouse) centralisant les donnees territoriales publiques de la region **Hauts-de-France** : schema en etoile deploye sur **Azure SQL**, six dimensions, sept tables de faits, pipelines ETL Python et infrastructure provisionnee via **Terraform**. Le projet E6 reprend cet entrepot et le fait evoluer en **systeme effectif en conditions reelles** : mise en place d'une methodologie de maintenance, journalisation et monitoring, strategie de sauvegarde, integration de nouvelles sources de donnees, gestion des acces, et implementation des variations de dimensions (Slowly Changing Dimensions) pour historiser les changements au fil du temps.

Pour ancrer le projet dans un contexte metier realiste, un **reseau d'agences bancaires fictif** a ete genere a partir du referentiel geographique de l'entrepot. En effet, les donnees reelles des employes d'une banque constituent des **donnees personnelles sensibles** (RGPD) qu'il est impossible d'importer sur une plateforme cloud Azure dans ce cadre. Les cent une agences ont donc ete positionnees sur les communes de la region dont la population depasse dix mille habitants, et un effectif fictif mais realiste d'environ cinq cent sept employes — directeur regional, directeurs de departement, directeurs d'agence et collaborateurs — a ete genere pour simuler la hierarchie d'un reseau bancaire regional.

Ce referentiel d'employes est le fondement du dispositif de **securite des acces** mis en oeuvre dans E6 : quatre roles orientes metier et une politique Row-Level Security (RLS) sur `dwh.dim_geographie` qui restreint automatiquement la vision des donnees de chaque consultant a son perimetre departemental. Ce rapport documente l'ensemble de ces evolutions et met a jour les modeles logique et physique de l'entrepot.

2 Methodologie et outillage de maintenance

2.1 Gestion des incidents

Tout incident est initie par la reception d'un message sur la **boite mail generique Outlook** de l'equipe data. Des la reception, un ticket est cree et le processus ci-dessous est applique.

1. **Creation du ticket d'incident**
   Des qu'un email est recu sur la boite generique de l'equipe, un ticket est ouvert avec les informations suivantes : date de reception, description de la demande, expediteur, impact presenti. Aucune demande ne doit rester sans ticket.
2. **Prise en charge**
   A reception du ticket, deux cas sont envisages :
   - **Un membre de l'equipe est disponible** → il s'attribue l'incident spontanement et s'en occupe.
   - **Aucun membre n'est disponible** → le **manager designe** la personne responsable du traitement.
3. **Classification et priorisation (avec le manager)**
   Le manager et le responsable designe determinent ensemble la priorite de la demande selon la matrice Impact × Urgence (cf. section 2.2) : **P1 Critique**, **P2 Haute**, **P3 Moyenne** ou **P4 Basse**. La priorite conditionne le delai de traitement attendu.
4. **Accuse de reception vers le metier**
   La personne en charge contacte le **demandeur ou le referent metier** pour :
   - confirmer que la demande est bien prise en compte,
   - communiquer la priorite attribuee et une estimation du delai de resolution.
5. **Traitement et diagnostic**
   Investigation du probleme : consultation des logs ETL (`dwh.log_etl`), analyse de la vue `analytics.v_monitoring_alertes`, verification des droits d'acces, identification du composant defaillant. Un journal de bord du diagnostic est maintenu dans le ticket.
6. **Resolution**
   Application du correctif adapte a la nature de l'incident : relance ETL, correction de donnees, ajustement de configuration, mise a jour des droits RLS, etc. La solution est testee et validee avant cloture.
7. **Notification de cloture au demandeur**
   La **personne qui a ouvert l'incident** est informee de la resolution par retour sur le fil mail ou via le ticket. La notification inclut : la cause identifiee, la solution appliquee et la date de cloture.
8. **Suivi post-incident et mise en backlog**
   Si l'incident revele un besoin d'evolution ou un risque recurrent, une tache est ajoutee au **backlog de l'equipe** en respectant la priorisation definie. Les incidents repetitifs font l'objet d'une analyse de cause racine pour eviter les recidives.

Flux simplifie : Mail recu → Ticket cree → Prise en charge → Priorite definie → Accuse metier → Diagnostic → Resolution → Notification → Backlog

2.2 Matrice de priorisation

La priorisation des incidents et des demandes de changement repose sur deux criteres : l'**impact** (combien d'utilisateurs ou de processus sont affectes) et l'**urgence** (a quelle vitesse le service doit etre retabli).

Impact / Urgence	Haute	Moyenne	Basse
Eleve (tous les utilisateurs)	P1 – Critique	P2 – Haute	P3 – Moyenne
Moyen (un service/equipe)	P2 – Haute	P3 – Moyenne	P4 – Basse
Faible (un utilisateur)	P3 – Moyenne	P4 – Basse	P4 – Basse

TABLE 1 – Matrice de priorisation Impact × Urgence

— **P1 – Critique** : Base de donnees inaccessible, perte de donnees. Resolution immediate (< 1h).

— **P2 – Haute** : ETL en echec, datamart non rafraichi. Resolution dans la journee (< 8h).

— **P3 – Moyenne** : Performance degradee, acces manquant pour un utilisateur. Resolution sous 48h.

— **P4 – Basse** : Demande d'evolution, amelioration cosmetique. Planification dans le sprint suivant.

3 Journalisation des alertes et des erreurs

3.1 Architecture de journalisation mise en place

La journalisation est le processus d'enregistrement chronologique des evenements qui se produisent dans un systeme informatiqu. Notre systeme de journalisation opere a deux niveaux complementaires :

3.1.1 Journalisation applicative (Python)

Les scripts ETL utilisent le module `logging` de Python pour produire des logs structures :

Code A.1 — *Configuration du logging dans les ETL* ⇒ voir Annexe 13, p. 25

3.1.2 Journalisation en base de donnees (SQL)

En complement des logs applicatifs, une table de journalisation SQL enregistre chaque etape de l'ETL directement dans l'entrepot :

```
CREATE TABLE dwh.log_etl (
    log_id          INT IDENTITY(1,1) PRIMARY KEY,
    date_execution  DATETIME DEFAULT GETDATE(),
    etape           NVARCHAR(100),   -- 'load_dimensions', 'load_facts'
        '...
    table_cible     NVARCHAR(100),   -- 'dwh.dim_temps', 'dwh.
       fait_revenus'
```

```
    statut            NVARCHAR(20),     -- 'DEBUT', 'SUCCES', 'ERREUR', '
        WARNING'
    nb_lignes         INT DEFAULT 0,
    duree_secondes    FLOAT DEFAULT 0,
    message           NVARCHAR(500),
    utilisateur       NVARCHAR(100) DEFAULT SYSTEM_USER
);
```

Listing 1 – Table de journalisation ETL

Point important : La double journalisation (fichier Python + table SQL) assure la tracabilite meme si l'un des deux systemes est indisponible. Les logs Python capturent les erreurs de connexion a la base, tandis que les logs SQL offrent une vue requetable pour les tableaux de bord de supervision.

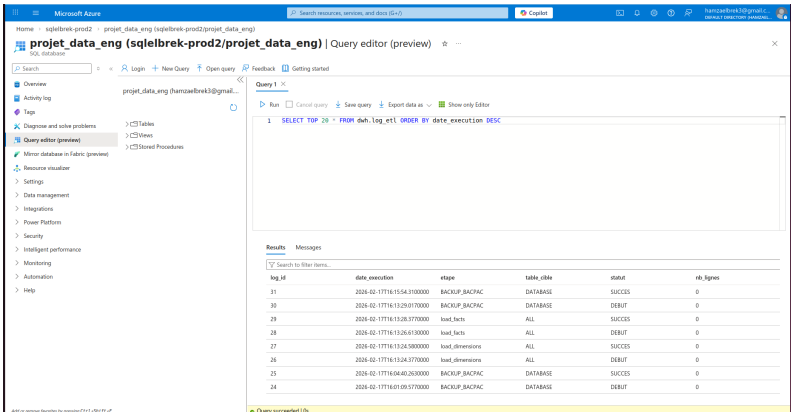

FIGURE 1 – Contenu de `dwh.log_etl` – Historique des 20 derniers evenements ETL (etape, statut, duree, nombre de lignes)

3.2 Fonction Python de journalisation en base

Code A.2 — *Fonction log_etl_db() dans les scripts ETL* ⇒ voir Annexe 14, p. 26

3.3 Vue de monitoring des alertes

Une vue analytique permet de superviser les alertes et erreurs depuis un outil BI :

```
CREATE VIEW analytics.v_monitoring_alertes AS
SELECT
    CAST(date_execution AS DATE) AS jour,
    etape,
    statut,
    COUNT(*) AS nb_evenements,
    SUM(CASE WHEN statut = 'ERREUR' THEN 1 ELSE 0 END) AS nb_erreurs,
    AVG(duree_secondes) AS duree_moyenne_sec
FROM dwh.log_etl
GROUP BY CAST(date_execution AS DATE), etape, statut;
```

Listing 2 – Vue de monitoring pour la supervision

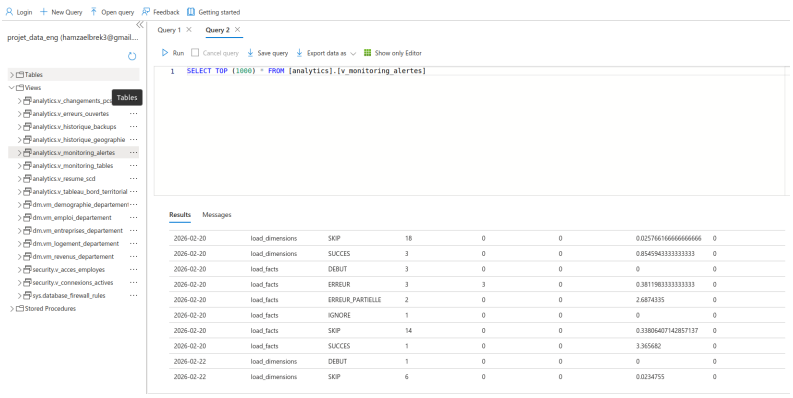

FIGURE 2 – Contenu de `dwh.log_erreurs` – Historique des 20 dernieres erreurs detectees durant les executions ETL

3.4 Notifications email automatiques du pipeline ETL

En complement de la journalisation en base, un module de notification email (`etl_notifier.py`) envoie automatiquement un rapport a la fin de chaque execution du pipeline ETL, qu'elle soit reussie ou en erreur.

3.4.1 Configuration

La configuration SMTP est stockee dans `terraform.tfvars` pour centraliser tous les parametres du projet. Les variables d'environnement ont priorite si elles sont definies :

```
# E6 - Notifications email ETL
etl_smtp_host     = "smtp.gmail.com"
etl_smtp_port     = "587"
etl_smtp_user     = "moncompte@gmail.com"
etl_smtp_password = "xxxx xxxx xxxx xxxx"   # App Password Gmail
etl_notify_email  = "destinataire@example.com"
```

Listing 3 – Configuration SMTP dans terraform.tfvars

Point important : Pour Gmail, il faut generer un **App Password** dans Securite → Mots de passe des applications (apres activation de la verification en 2 etapes). Ce mot de passe a 16 caracteres remplace le mot de passe Google dans la configuration ETL.

3.4.2 Email de succes – fin de pipeline

A la fin d'un pipeline complet sans erreur, l'orchestrateur (`run_etl.py`) appelle `send_success_email` qui envoie un rapport HTML structure :

```python
# En cas de succes global du pipeline
if success:
    rapport_etapes['details'] = rapport_details
    send_success_email(rapport_etapes, smtp_config=smtp_config)
```

Listing 4 – Declenchement de la notification dans run_etl.py

L'email de succes contient deux tableaux :

— **Recapitulatif des etapes** : statut (OK/ERREUR), nombre de lignes et duree pour chacune des 5 etapes (Staging, Dimensions, Faits, Refresh, Backup)

— **Detail par table** : statut (OK, SKIP, IGNORE, ERREUR), heure et duree pour chaque table chargee (dimensions + faits)

3.4.3 Email d'alerte – erreur en cours de pipeline

Des qu'une erreur est detectee sur une table ou une etape, l'orchestrateur envoie immediatement un email d'alerte sans attendre la fin du pipeline :

```python
send_error_email(
    etape='Faits',
    table='dwh.fait_revenus',
    erreur="KeyError: colonne 'obs_value' introuvable",
    heure='14:32:18',
    rapport_partiel={   # Tables deja traitees avant l'erreur
        'dwh.fait_population': {'statut': 'OK', 'nb_lignes': 45},
        'dwh.fait_entreprises': {'statut': 'SKIP', 'nb_lignes': 0},
    },
    smtp_config=smtp_config,
)
```

Listing 5 – Envoi d'alerte en cas d'erreur ETL (run_etl.py)

L'email d'alerte contient :

— Un bloc rouge avec : l'etape, la table en erreur, l'heure et le message d'erreur complet

— Un tableau des tables deja traitees avant l'erreur (avec leurs statuts)

3.4.4 Statuts des tables dans les emails

Statut	Icone	Signification
OK	$\surd$	Table chargee avec succes
SKIP	$\rightarrow$	Table ignoree (deja a jour, pas de nouvelles donnees)
IGNORE	$\rightarrow$	Table ignoree (table staging source manquante)
ERREUR	$\times$	Erreur lors du chargement

TABLE 2 – Statuts des tables dans les emails de notification ETL

Explication technique : Architecture de notification

Le module `etl_notifier.py` utilise uniquement la bibliotheque standard Python (`smtplib`, `email.mime`) – aucune dependance externe necessaire. La connexion SMTP est etablie en **TLS** (port 587) avec `STARTTLS`. Si la configuration SMTP est absente ou incomplete, les notifications sont simplement ignorees (log `WARNING`) sans bloquer le pipeline.

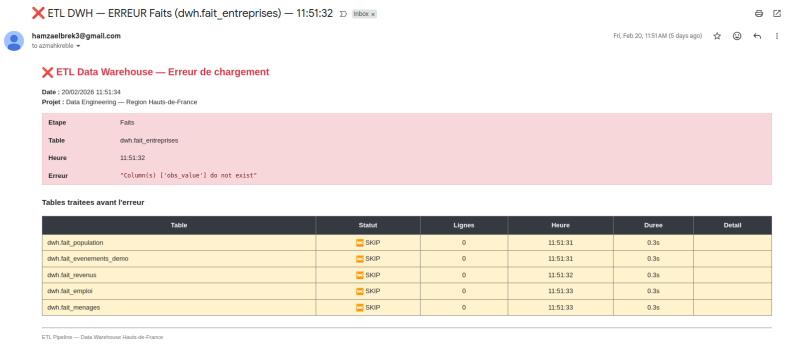

FIGURE 3 – Exemple de notification email automatique – Rapport de fin de pipeline ETL (succes ou erreur)

4 Strategie de sauvegarde et restauration

4.1 Niveau 1 : Sauvegardes automatiques Azure (PITR)

Azure SQL Database effectue automatiquement des sauvegardes, sans aucune intervention de notre part. Ce mecanisme, appele **PITR** (Point-In-Time Restore), constitue le premier niveau de notre strategie.

La politique de retention est definie dans notre infrastructure Terraform :

```
resource "azurerm_mssql_database" "sql" {
  # ...
  short_term_retention_policy {
    retention_days           = 14    # PITR sur 14 jours
    backup_interval_in_hours = 12    # Differentiel toutes les 12h
  }

  long_term_retention_policy {
    weekly_retention  = "P4W"    # 4 semaines
    monthly_retention = "P12M"   # 12 mois
    yearly_retention  = "P3Y"    # 3 ans
    week_of_year      = 1
  }
}
```

Listing 6 – Configuration Terraform de la retention des sauvegardes (main.tf)

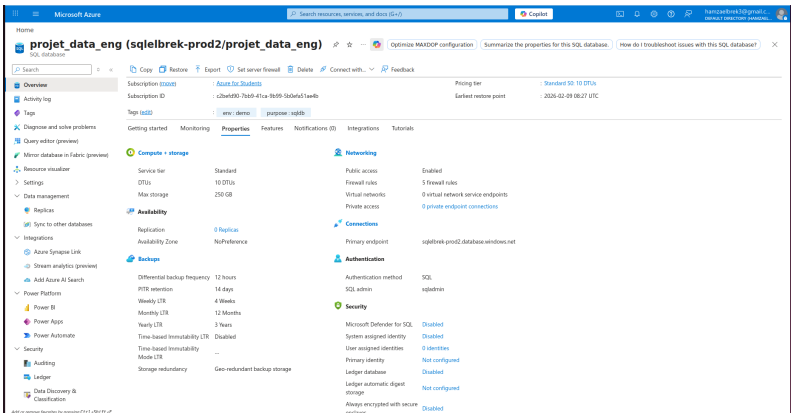

FIGURE 4 – Configuration des sauvegardes dans le portail Azure – Politique PITR (14 jours) et retention long terme (4 semaines, 12 mois, 3 ans)

4.2 Niveau 2 : Recuperation et export BACPAC vers le Data Lake

Les sauvegardes automatiques Azure sont gerees par la plateforme et restent dans l'ecosysteme Azure. Pour disposer d'une copie **independante** et **portable** de notre base, nous ajoutons un second niveau : apres chaque execution de l'ETL, un script Python recupere l'etat de la base et l'exporte en fichier `.bacpac` horodate vers notre Data Lake ADLS Gen2.

Le principe est le suivant :

1. Azure realise automatiquement ses backups (complet + differentiel + logs de transactions)
2. Notre script `backup_to_datalake.py`, integre comme **etape 5 du pipeline ETL**, exporte la base via `az sql db export` vers le Data Lake
3. Le fichier `.bacpac` obtenu est stocke dans le container `raw/backups/` avec un horodatage
4. Ce fichier peut etre utilise a tout moment pour restaurer la base via `az sql db import`

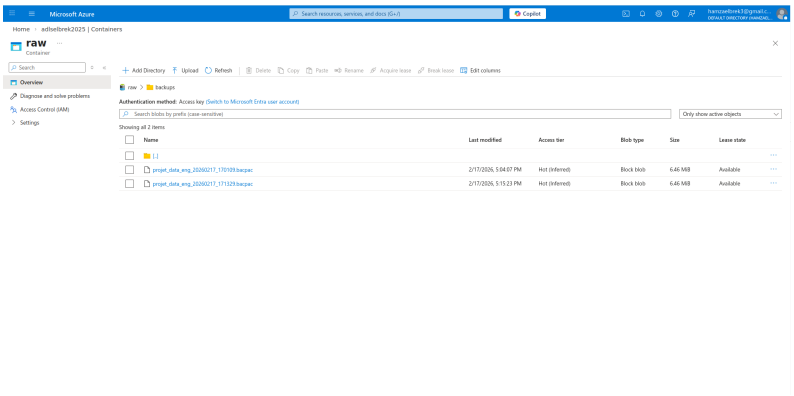

FIGURE 5 – Fichiers `.bacpac` stockes dans le Data Lake – Container `raw/backups/` avec horodatage

Code A.3 — *Flux de la sauvegarde* ⇒ voir Annexe 15, p. 26

4.2.1 Script d'export BACPAC

Code A.4 — *Export BACPAC vers le Data Lake (backup_to_datalake.py)*⇒ voir Annexe 16, p. 26

4.2.2 Nettoyage automatique des anciens backups

Le script gere egalement la suppression des fichiers `.bacpac` de plus de 30 jours (configurable) pour eviter l'accumulation dans le Data Lake :

Code A.5 — *Nettoyage automatique des anciens backups* ⇒ voir Annexe 17, p. 27

4.3 Restauration de la base de donnees

En cas de besoin (corruption, perte de donnees, incident), deux methodes de restauration sont disponibles selon la situation :

- **Restauration Point-in-Time (PITR)** : Utilise les sauvegardes automatiques Azure pour restaurer la base a n'importe quel instant dans la fenetre de retention (14 jours). Methode la plus rapide, ideale pour les incidents recents.
- **Import BACPAC depuis le Data Lake** : Recree la base a partir d'un fichier `.bacpac` exporte par notre pipeline ETL. Utile pour restaurer un etat au-dela de la fenetre PITR, ou pour migrer la base vers un autre serveur.

Code A.6 — *Restauration via Azure CLI* ⇒ voir Annexe 18, p. 27

4.4 Monitoring des sauvegardes

Deux vues SQL permettent de surveiller l'etat des sauvegardes :

- `analytics.v_etat_backup_azure` : Interroge la DMV `sys.dm_database_backups` pour afficher les sauvegardes automatiques Azure (type, date, taille).
- `analytics.v_historique_backups` : Filtre la table `dwh.log_etl` pour afficher l'historique des exports BACPAC effectues par le pipeline ETL.

```
CREATE VIEW analytics.v_historique_backups AS
SELECT
    log_id, date_execution,
    etape AS type_backup, statut, message,
    duree_secondes, nb_lignes, utilisateur
FROM dwh.log_etl
WHERE etape IN ('BACKUP_BACPAC', 'RESTAURATION');
```

Listing 7 – Vue de l'historique des exports BACPAC

5 Integration de nouvelles sources de donnees

L'entrepot E5 ne chargeait que 4 tables de faits sur les 7 prevues. Pour E6, deux nouvelles sources CSV ont ete integrees : `EMPLOI_CHOMAGE_hauts_de_france.csv` alimentant `fait_emploi`, et `Menage_hauts_de_france.csv` alimentant `fait_menages`. Le processus ETL suit un pattern uniforme : les donnees sont d'abord chargees depuis la zone de staging, les cles etrangeres sont resolues via les dimensions `dim_temps` et `dim_geographie`, les valeurs sont agregees par annee et departement, puis inserees dans la table de faits. Chaque chargement est trace dans le journal `log_etl` en base de donnees.

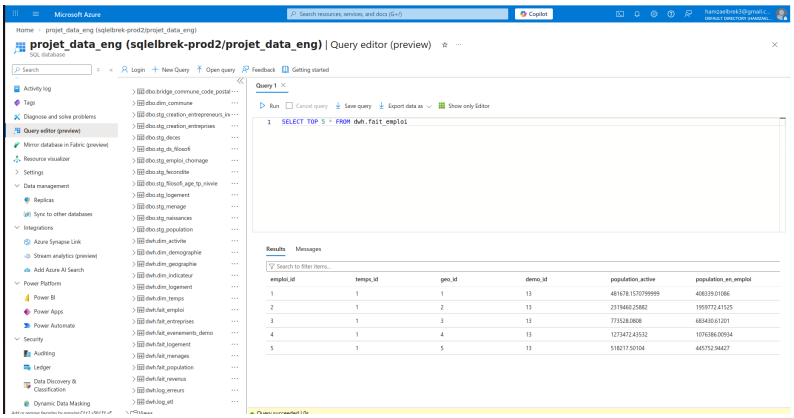

FIGURE 6 – Donnees chargees dans `dwh.fait_emploi` – Exemple des 5 premieres lignes apres integration de la nouvelle source

6 Gestion des acces et securite

La gestion des acces constitue un pilier fondamental de la gouvernance de l'entrepot de donnees. Cette section decrit l'evolution du dispositif de securite entre le projet initial (E5) et la version actuelle (E6), ainsi que les procedures operationnelles associees.

6.1 Roles initiaux du projet E5

Dans le projet initial E5, l'entrepot de donnees disposait de trois roles RBAC (*Role-Based Access Control*) couvrant les grands profils d'utilisation :

Role	Permissions	Utilisateurs types
`role_etl_process`	Lecture/ecriture sur `stg` et `dwh`, lecture sur `dm` et `analytics`	Comptes de service ETL
`role_analyst`	Lecture complete sur `dwh` (dimensions et faits), `dm` et `analytics`. Aucun acces au staging.	Data Analysts
`role_dwh_admin`	Controle total sur tous les schemas et objets	Administrateurs DBA

TABLE 3 – Roles RBAC du projet initial E5

Ce dispositif repondait correctement aux besoins d'un entrepot centralise avec des profils generiques. Cependant, lors du passage en production avec les 519 collaborateurs repartis sur les 5 departements de la region Hauts-de-France, une **limite structurelle** est apparue : ces roles ne prenaient pas en compte les restrictions geographiques.

En effet, un collaborateur de l'agence de Valenciennes (Nord, dept. 59) disposant du role `role_analyst` avait techniquement acces aux donnees de toute la region — y compris les departements de l'Oise (60) et de la Somme (80) dans lesquels il n'a aucune responsabilite. Cette situation est incompatible avec le principe RGPD de *minimisation des acces*.

6.2 Refonte des roles lors de l'evolution E6

Pour repondre a cette limite, l'evolution E6 a redesigne le dispositif de securite selon deux axes complementaires :

1. **Refonte des roles RBAC** : suppression du role `role_dwh_admin`, scission du role `role_analyst` en deux profils distincts, et ajout d'un role `role_admin` avec permissions explicites.
2. **Introduction du RLS** (*Row-Level Security*) : filtrage automatique des donnees par zone geographique pour les collaborateurs en agence.

Role	Permissions	Utilisateurs types
`role_admin`	Controle total (`CONTROL`) sur tous les schemas. Peut creer des tables, vues, procedures.	Administrateurs DBA
`role_etl_process`	Lecture/ecriture (`SELECT, INSERT, UPDATE, DELETE`) sur `stg` et `dwh`. Lecture seule sur `dm` et `analytics`.	Comptes de service ETL
`role_analyst`	Lecture seule sur `dwh` (dimensions + faits), `dm` et `analytics`. Aucun acces au staging.	Data Analysts internes
`role_consultant`	Lecture seule sur `dm` et `analytics` uniquement. **Acces filtre par departement via RLS** (voir section 6.4).	Collaborateurs agences

Table 4 – Roles RBAC refondus lors de l'evolution E6

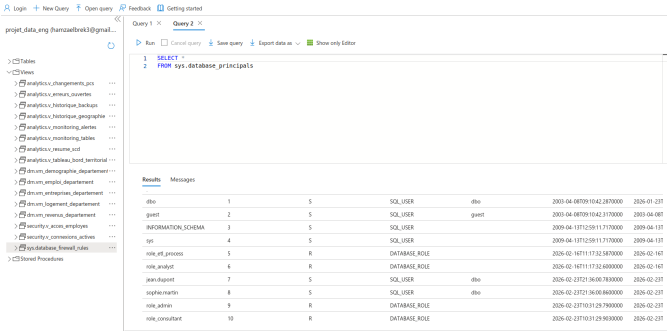

FIGURE 7 – Roles de base de donnees crees dans Azure SQL – Resultat de `SELECT name FROM sys.database_principals WHERE type = 'R'`

6.3 Procedure d'integration d'un nouveau collaborateur

Lorsqu'un nouveau collaborateur rejoint la structure, l'integration de ses acces a l'entrepot suit un processus en trois etapes selon son profil.

Etape 1 — Determiner le role approprie

Profil	Role a attribuer
Administrateur / DBA	`role_admin`
Ingenieur ETL / pipeline de donnees	`role_etl_process`
Data Analyst (acces region entiere)	`role_analyst`
Collaborateur en agence (acces departement)	`role_consultant`

TABLE 5 – Correspondance profil metier / role SQL

Etape 2 — Creer l'utilisateur et attribuer le role

```sql
-- Creer l'utilisateur avec mot de passe (Azure SQL Database)
CREATE USER [prenom.nom] WITH PASSWORD = 'MotDePasseSecurise!';

-- Attribuer le role correspondant au profil
ALTER ROLE role_consultant ADD MEMBER [prenom.nom];
```

Listing 8 – Creation d'un nouvel acces utilisateur dans Azure SQL

Etape 3 — Configurer la zone geographique (si role_consultant)

Pour un collaborateur en agence, il faut enregistrer son perimetre geographique dans la table `security.utilisateurs_zones` afin que le RLS filtre correctement ses acces :

```sql
-- Acces restreint a un departement (ex : Nord - 59)
INSERT INTO security.utilisateurs_zones (login_sql, departement_code)
VALUES ('prenom.nom', '59');

-- Acces region entiere (ex : directeur regional)
INSERT INTO security.utilisateurs_zones (login_sql, departement_code)
VALUES ('prenom.nom', NULL);
```

Listing 9 – Enregistrement de la zone geographique du collaborateur

Explication technique : Logique du filtre RLS

La fonction predicat `security.fn_rls_geographie` verifie la presence du login dans `utilisateurs_zones`. Si le collaborateur n'y figure pas, il a acces a toutes les lignes (cas des roles `role_admin`, `role_analyst`, `role_etl_process`). S'il y figure avec un `departement_code` non null, il ne voit que les donnees de ce departement. Avec `NULL`, il voit toute la region.

6.4 Row-Level Security (RLS) pour les consultants

6.4.1 Architecture RLS mise en place

Le RLS s'applique sur la table `dwh.dim_geographie` (qui contient le `departement_code`). Puisque toutes les vues des schemas `dm` et `analytics` effectuent une jointure sur cette table, le filtre se propage automatiquement a toutes ces vues sans modification supplementaire.

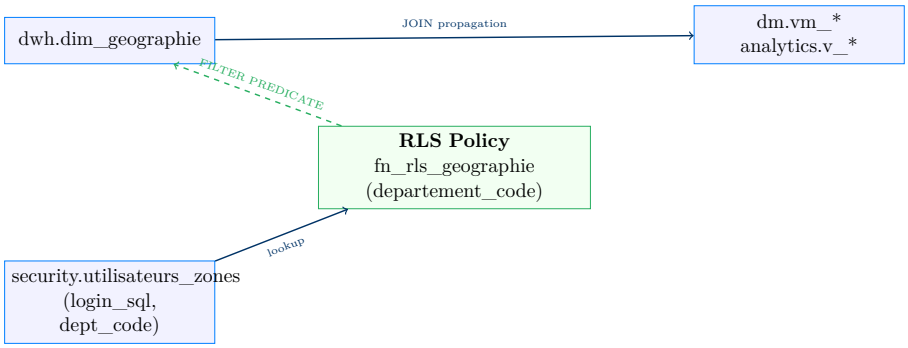

FIGURE 8 – Architecture RLS – Filtre sur dim_geographie, propagation aux datamarts

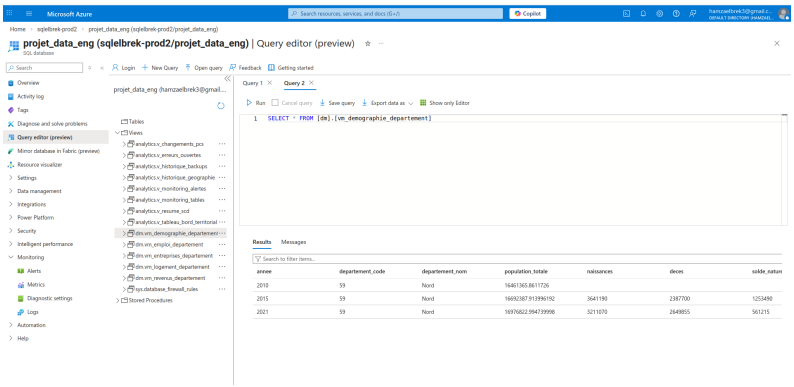

FIGURE 9 – Verification du filtre RLS en base – Un consultant du departement 59 ne voit que les donnees de son perimetre

6.4.2 Schema de securite et tables

Un schema `security` dedie contient les trois tables du dispositif RLS :

Code A.7 — *Tables du schema security (011_security_rls.sql)* ⇒ voir Annexe 19, p. 28

6.4.3 Hierarchie des employes

La structure hierarchique compte 4 niveaux pour les 5 departements de la region (02 Aisne, 59 Nord, 60 Oise, 62 Pas-de-Calais, 80 Somme) :

Niveau	Effectif	Perimetre RLS
`DIRECTEUR_REGIONAL`	1	Region entiere (5 departements)
`DIRECTEUR_DEPARTEMENT`	5	Leur departement
`DIRECTEUR_AGENCE`	101	Departement de leur agence
`COLLABORATEUR`	412	Departement de leur agence
Total		**519 employes**

TABLE 6 – Hierarchie des employes et perimetre d'acces RLS

Les 101 agences correspondent aux communes de la region dont la population depasse 10 000 habitants (9 grandes agences >50 000 hab., 41 moyennes, 51 petites). Le script Python `analytics/etl/load_security.py` genere automatiquement cette structure a partir du referentiel geographique `communes.json`.

6.4.4 Fonction predicat et politique de securite

Code A.8 — *Fonction predicat RLS (011_security_rls.sql)* ⇒ voir Annexe 20, p. 28

Explication technique : Propagation automatique du filtre

Le predicat RLS est applique sur `dwh.dim_geographie`. Toutes les vues `dm.*` et `analytics.*` joignent cette table via `geo_id`. Lorsqu'un consultant execute une requete sur `dm.vm_demographie_departement`, SQL Server filtre d'abord les lignes de `dim_geographie` selon son predicat, puis effectue la jointure avec les tables de faits. Le resultat est automatiquement restreint a ses departements autorises, sans modifier les vues ni les requetes.

6.4.5 Script ETL de chargement du referentiel securite

Le script `analytics/etl/load_security.py` permet de generer et charger les donnees de securite :

```
# Verification sans chargement en base
python load_security.py --check

# Chargement initial (ou rechargement complet)
python load_security.py --reset --load

# Resultat :
```

```
# 101 agences  -> security.agences
# 519 employes -> security.employes
# 519 zones    -> security.utilisateurs_zones
```

Listing 10 – Utilisation de load_security.py

7 Procedures d'evolution et de scalabilite

L'entrepot de donnees est concu pour evoluer sans interruption de service. Quatre procedures operationnelles encadrent les operations courantes d'administration :

- **Procedure 1** : Creer un nouvel acces utilisateur
- **Procedure 2** : Ajouter un nouveau datamart analytique
- **Procedure 3** : Montee en charge (scalabilite)
- **Procedure 4** : Ajouter une nouvelle source de donnees

Le detail complet de chaque procedure est documente en annexe :

Annexe B — *Procedures d'exploitation et de scalabilite* ⇒ voir Annexe A.1, p. 24

8 Variations de dimensions (Slowly Changing Dimensions)

8.1 SCD Type 1 : Ecrasement (Overwrite)

Definition : SCD Type 1

Le Type 1 consiste a **ecraser purement et simplement l'ancienne valeur** par la nouvelle. L'historique est perdu.

Exemple : Le libelle de la section NAF "C" est corrige de "Industrie manufacturere" (faute) a "Industrie manufacturiere".

Code A.10 — *Implementation SCD Type 1 – Ecrasement* ⇒ voir Annexe 21, p. 29

8.2 SCD Type 2 : Historisation (Add New Row)

Definition : SCD Type 2

Le Type 2 est l'approche la plus riche : au lieu d'ecraser l'ancienne valeur, on **cree une nouvelle ligne** pour la nouvelle valeur et on **ferme l'ancienne ligne** en marquant sa date de fin. Chaque version de la dimension coexiste dans la table avec des colonnes de gestion :

- `date_debut_validite` : Date a partir de laquelle cette version est valide
- `date_fin_validite` : Date de fin de validite (NULL = version courante)
- `est_actif` : Booleen indiquant si c'est la version en vigueur
- `version` : Numero de version incremental

Exemple : La commune "Noyelles-Godault" (code 62617) fusionne avec "Henin-Beaumont" en 2026. On conserve l'historique des deux noms.

```sql
-- Ajout des colonnes SCD Type 2 a dim_geographie
ALTER TABLE dwh.dim_geographie ADD
    date_debut_validite DATETIME DEFAULT '2010-01-01',
    date_fin_validite   DATETIME NULL,   -- NULL = actif
    est_actif           BIT DEFAULT 1,
    version             INT DEFAULT 1;
```

Listing 11 – Colonnes SCD Type 2 ajoutees a dim_geographie

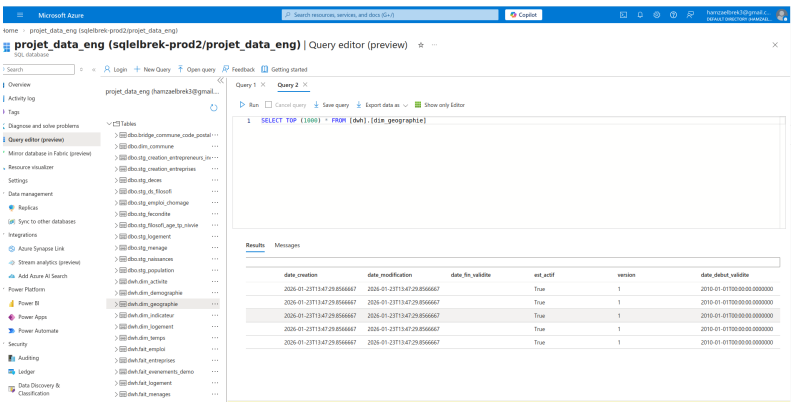

FIGURE 10 – Colonnes SCD Type 2 dans `dwh.dim_geographie` – Champs `est_actif`, `version` et `date_debut_validite` pour l'historisation des communes

Code A.11 — *Procedure SCD Type 2 – Historisation* ⇒ voir Annexe 22, p. 30

8.3 SCD Type 3 : Colonne precedente (Add New Column)

Definition : SCD Type 3

Le Type 3 est un compromis entre le Type 1 et le Type 2 : on ajoute une **colonne supplementaire** pour stocker l'ancienne valeur, tout en mettant a jour la valeur courante. Cette approche ne conserve qu'**un seul niveau d'historique** (la valeur precedente).

- `libelle_actuel` : La valeur courante de l'attribut
- `libelle_precedent` : L'ancienne valeur (avant le dernier changement)
- `date_changement` : La date du dernier changement

Exemple : La categorie PCS "Cadres" est renommee en "Cadres et professions intellectuelles superieures".

```sql
-- Ajout des colonnes SCD Type 3 a dim_demographie
ALTER TABLE dwh.dim_demographie ADD
    ancien_libelle_pcs  NVARCHAR(200) NULL,
    date_changement_pcs DATETIME NULL;
```

Listing 12 – Colonnes SCD Type 3 ajoutees a dim_demographie

Code A.12 — *Procedure SCD Type 3* ⇒ voir Annexe 23, p. 30

9 Documentation des evolutions et modeles mis a jour

9.1 Evolution des Architectures de Data Warehouse

Le modele logique E6 s'articule desormais autour de **cinq schemas** avec des responsabilites clairement separees :

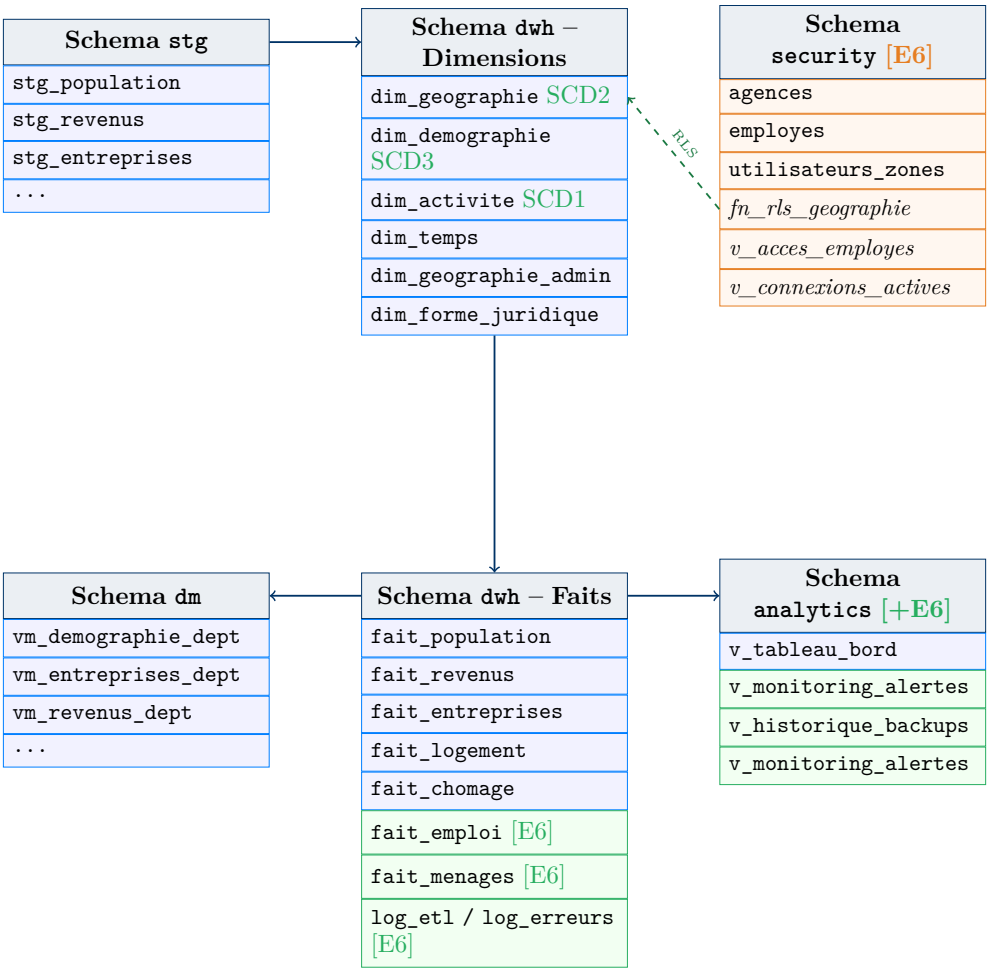

FIGURE 11 – Modele logique E6 – Vue globale des 5 schemas (vert = nouveau E6, orange = schema security)

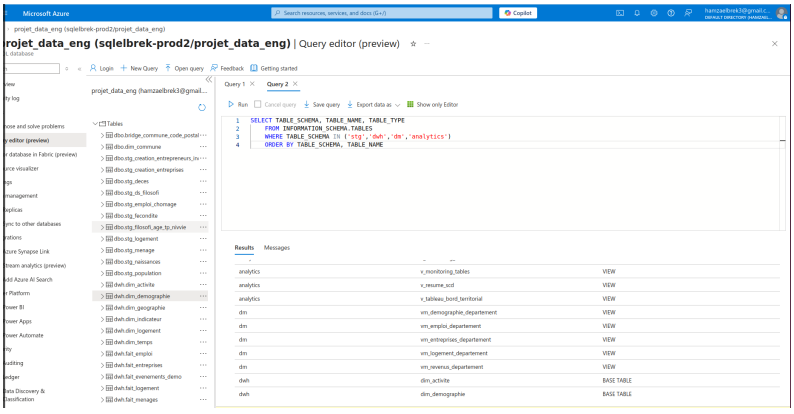

FIGURE 12 – Schemas et tables de l'entrepot dans Azure SQL – Vue `INFORMATION_SCHEMA.TABLES` listant tous les objets des schemas `stg`, `dwh`, `dm` et `analytics`

9.2 Nouveau modele logique (E6)

Le schema en etoile complet de l'entrepot apres evolution E6 regroupe **6 dimensions** et **7 tables de faits** dans le schema `dwh`.

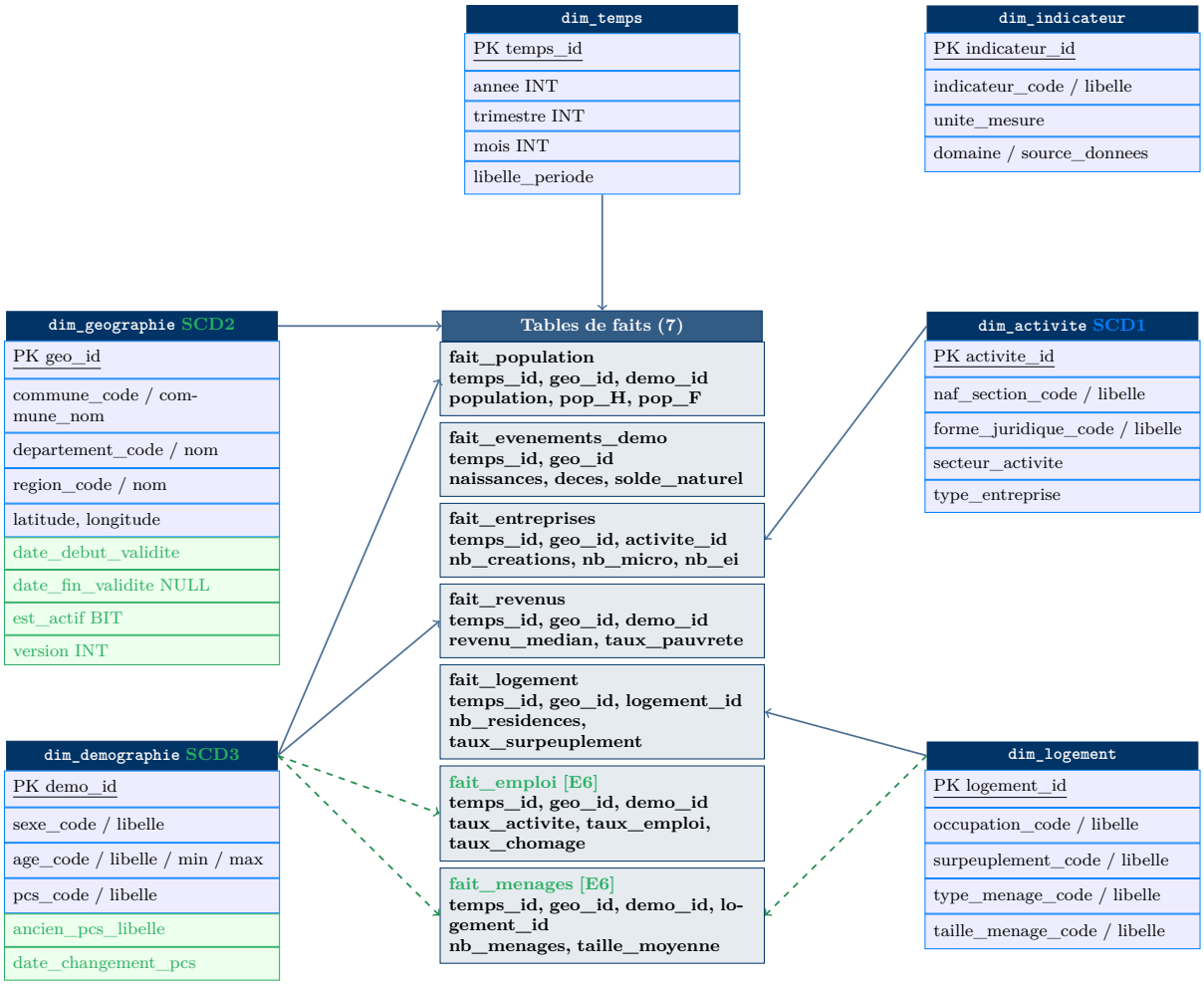

FIGURE 13 – Schema en etoile complet – Data Warehouse E6 (vert = nouveautes E6, tirete = nouvelles liaisons)

9.3 Nouveau modele physique (E6)

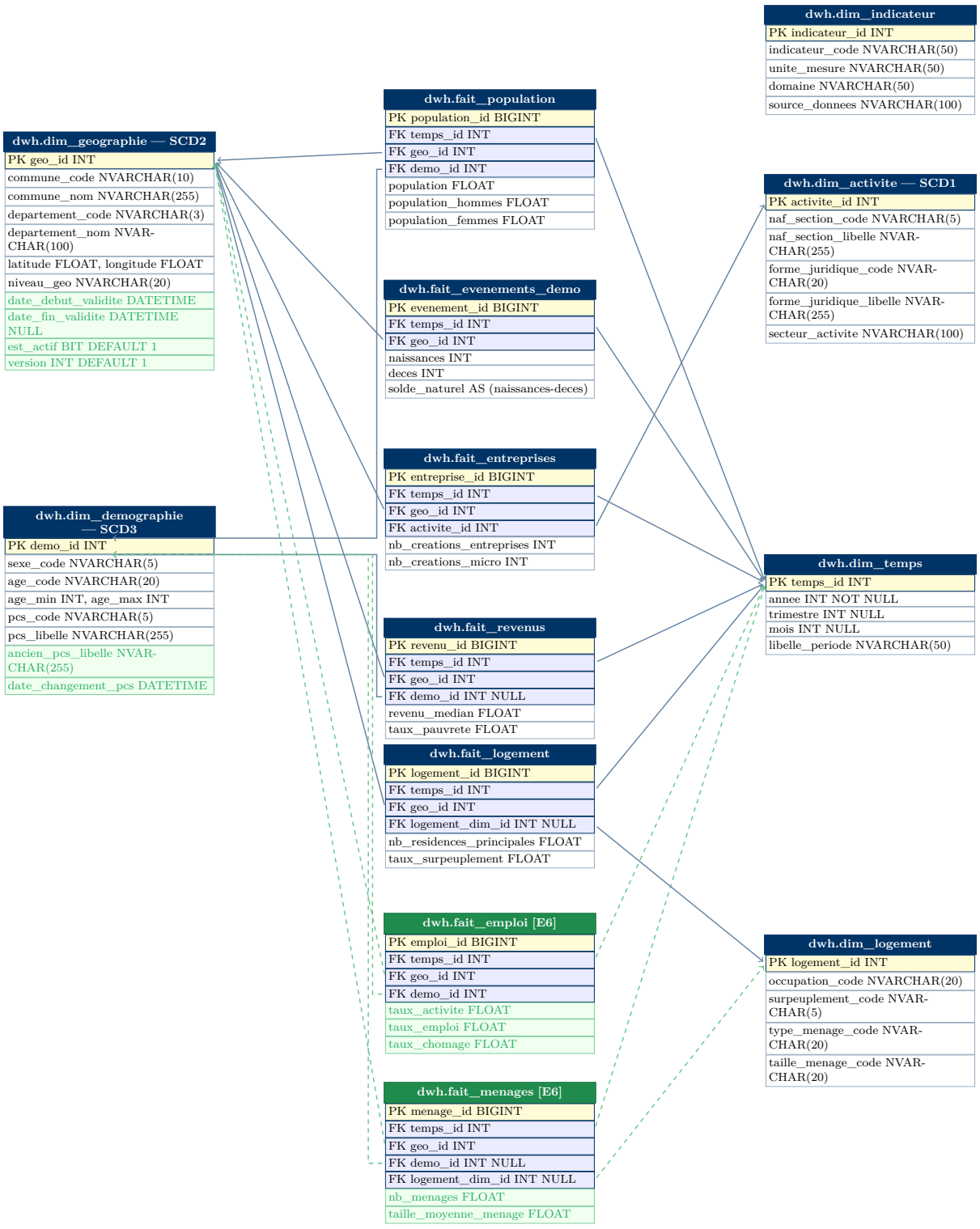

FIGURE 14 – Modele physique ERD – Data Warehouse E6 complet (jaune = PK, bleu clair = FK, vert = nouveautes E6, fleche pleine = lien E5, fleche tiretee = nouveau lien E6)

10 Conclusion

Le projet E6 a transforme l'entrepot de donnees E5 en un systeme operationnel complet. Sur le plan de la maintenance, un processus de gestion des incidents calque sur la realite de l'equipe a ete formalise, accompagne d'une journalisation a deux niveaux (logs Python et table `dwh.log_etl`) et de notifications email automatiques a l'issue de chaque execution du pipeline. Une strategie de sauvegarde avec export BACPAC vers le Data Lake ADLS Gen2 et des politiques de retention Azure completent ce dispositif. L'entrepot a egalement ete etendu par l'integration de deux nouvelles tables de faits (emploi, menages) et la refonte des roles de securite : quatre roles RBAC metier et un mecanisme de Row-Level Security sur `dwh.dim_geographie` qui restreint automatiquement les donnees visibles par chaque consultant a son perimetre departemental.

Sur le plan dimensionnel, les trois types de variations de Kimball (SCD 1, 2 et 3) ont ete implementes sur des dimensions distinctes, avec les procedures SQL `MERGE` et les adaptations ETL correspondantes. La simulation d'un effectif fictif de cinq cent sept employes repartis dans cent une agences ancrees dans la geographie regionale a permis de valider concretement le dispositif de securite dans un contexte metier realiste. L'ensemble des evolutions est documente dans ce rapport avec la mise a jour des modeles logique et physique de l'entrepot.

Pour aller plus loin, plusieurs evolutions pourraient prolonger ce travail : integration d'Azure Monitor pour des alertes temps reel, mise en place d'un Change Data Capture (CDC) pour declencher les ETL a la source, adoption d'un framework de qualite des donnees (Great Expectations, dbt), ou migration du pipeline vers Azure Data Factory pour une orchestration native et un monitoring centralise.

A Annexes

A.1 Procedures d'exploitation et de scalabilite

Procedure 1 – Creer un nouvel acces utilisateur

1. Recevoir la demande d'acces via le systeme de ticketing (type : `changement`, priorite : P4)
2. Identifier le role approprie selon le profil : `role_analyst` (analyste interne) ou `role_consultant` (collaborateur agence avec RLS)
3. Verifier que l'employe existe dans `security.employes` ; si absent, l'ajouter avec le script `load_security.py --load`
4. Creer l'utilisateur SQL et assigner le role via le script ETL : `python load_security.py --create-users`
5. Verifier les permissions effectives avec `security.v_acces_employes` et tester la connexion depuis Azure Data Studio
6. Documenter dans le ticket : login cree, role attribue, zone RLS, date
7. Notifier l'utilisateur de ses identifiants par canal securise

Procedure 2 – Ajouter un nouveau datamart analytique

1. Recevoir le besoin d'analyse (ticket type : `changement`)
2. Identifier les tables de faits et dimensions impliquees
3. Creer la vue dans le schema `dm` en jointurant sur les dimensions actives (`est_actif = 1` si SCD Type 2 implique)
4. Accorder les droits de lecture : `GRANT SELECT ON dm.<vue> TO role_consultant`
5. Valider la propagation du filtre RLS si la vue jointure sur `dim_geographie`
6. Ajouter un test dans `test_e6_evolution.py` pour verifier l'existence de la vue
7. Documenter la vue dans le dictionnaire de donnees et mettre a jour le modele logique

Code A.9 — *Exemple de creation de vue datamart* ⇒ voir Annexe A.2, p. 29

Procedure 3 – Montee en charge (scalabilite)

1. **Diagnostic** : Identifier le goulot d'etranglement via les vues de monitoring (`analytics.v_resume_et` `sys.dm_exec_sessions`)
2. **Base de donnees** : Augmenter le SKU Azure SQL dans `terraform.tfvars` (ex : S0 → S2 → S4) puis `terraform apply`
3. **Partitionnement** : Si les tables de faits depassent 10 millions de lignes, envisager un partitionnement par annee sur `annee_id`
4. **Index** : Ajouter des index couvrants sur les colonnes de filtrage frequentes (departement_code, annee_id)
5. **ETL** : Activer le chargement parallele par departement dans `run_etl.py` pour reduire la duree du pipeline

6. **Archivage** : Exporter les donnees anterieures a N-3 vers le Data Lake via `backup_to_datalake.py` et les supprimer du DWH actif
7. **Validation** : Controler les performances apres chaque modification et documenter les gains dans le ticket de changement

Procedure 4 – Ajouter une nouvelle source de donnees

1. **Analyse** : Etudier le format (CSV, API, JSON), le volume et la frequence de mise a jour de la nouvelle source
2. **Staging** : Creer la table de staging `stg.stg_nouvelle_source` avec les colonnes brutes de la source
3. **Mapping** : Identifier les correspondances avec les dimensions existantes (ou creer de nouvelles dimensions si necessaire)
4. **Schema** : Creer ou modifier la table de faits dans le schema `dwh` avec les cles etrangeres vers les dimensions
5. **ETL** : Developper la fonction de chargement dans `load_facts.py` en respectant le pattern SCD adequat pour chaque dimension impliquee
6. **Datamart** : Creer ou mettre a jour les vues dans le schema `dm` (cf. Procedure 2)
7. **Tests** : Ajouter les tests de validation dans `test_e6_evolution.py`
8. **Deploiement** : Integrer le script SQL dans le pipeline Terraform (`Terraform/sql/`) et valider avec `terraform apply`
9. **Documentation** : Mettre a jour le modele logique et le dictionnaire de donnees

A.2 Scripts et extraits de code

A.1 — Configuration du logging dans les ETL

```python
import logging

# Configuration du logger avec sortie fichier + console
logging.basicConfig(
    level=logging.INFO,
    format='%(asctime)s [%(levelname)s] %(name)s - %(message)s',
    handlers=[
        logging.FileHandler('etl_pipeline.log'),
        logging.StreamHandler()  # Sortie console
    ]
)
logger = logging.getLogger('etl_pipeline')

# Exemples d'utilisation
logger.info("Debut du chargement de dim_geographie")
logger.warning("Table staging vide, etape ignoree")
logger.error("Cle etrangere invalide pour fait_revenus")
```

Listing 13 – Configuration du logging dans les ETL

A.2 — Fonction log_etl_db() dans les scripts ETL

```
def log_etl_db(engine, etape, table_cible, statut,
               nb_lignes=0, duree=0, message=''):
    """Enregistre un evenement ETL dans dwh.log_etl."""
    try:
        with engine.connect() as conn:
            conn.execute(text("""
                INSERT INTO dwh.log_etl
                  (etape, table_cible, statut, nb_lignes,
                   duree_secondes, message)
                VALUES (:etape, :table, :statut, :nb,
                        :duree, :msg)
            """), {
                'etape': etape, 'table': table_cible,
                'statut': statut, 'nb': nb_lignes,
                'duree': duree, 'msg': message
            })
            conn.commit()
    except Exception as e:
        logger.error(f"Impossible de loguer en BDD: {e}")
```

Listing 14 – Fonction log_etl_db() dans les scripts ETL

A.3 — Flux de la sauvegarde

```
Azure SQL Database
  |
  |  Sauvegardes automatiques (complet + differentiel + logs)
  |  gerees par Azure (PITR, retention 14 jours)
  |
  v
backup_to_datalake.py (etape 5 du pipeline ETL)
  |
  |  az sql db export  -->  fichier .bacpac
  |
  v
Data Lake ADLS Gen2
  raw/backups/projet_data_eng_20260217_143000.bacpac
  |
  +-- Log dans dwh.log_etl
  +-- Nettoyage automatique des .bacpac > 30 jours
```

Listing 15 – Flux de la sauvegarde

A.4 — Export BACPAC vers le Data Lake (backup_to_datalake.py)

```
def export_bacpac(config, resource_group):
    """Exporte la base Azure SQL en .bacpac via az sql db export."""
    storage_account = config['storage_account']
    timestamp = datetime.now().strftime('%Y%m%d_%H%M%S')
    blob_name = f"backups/{config['database']}_{timestamp}.bacpac"

    # Recuperer la cle du storage account
    result = subprocess.run(
        ['az', 'storage', 'account', 'keys', 'list',
```

```
        '--account-name', storage_account,
        '--resource-group', resource_group,
        '--query', '[0].value', '-o', 'tsv'],
       capture_output=True, text=True)
   storage_key = result.stdout.strip()

   storage_uri = (f"https://{storage_account}.blob.core.windows.net"
                  f"/raw/{blob_name}")

   # Exporter la base en .bacpac directement vers le blob
   subprocess.run([
       'az', 'sql', 'db', 'export',
       '--admin-user', config['user'],
       '--admin-password', config['password'],
       '--name', config['database'],
       '--server', config['server'],
       '--resource-group', resource_group,
       '--storage-key', storage_key,
       '--storage-key-type', 'StorageAccessKey',
       '--storage-uri', storage_uri,
   ], capture_output=True, text=True, timeout=1800)

   return blob_name
```

Listing 16 – Export BACPAC vers le Data Lake (backup_to_datalake.py)

A.5 — Nettoyage automatique des anciens backups

```
def cleanup_old_backups(config, resource_group, retention_days=30):
    """Supprime les .bacpac de plus de retention_days jours."""
    result = subprocess.run(
        ['az', 'storage', 'blob', 'list',
         '--account-name', config['storage_account'],
         '--container-name', 'raw', '--prefix', 'backups/',
         '--query', '[].{name:name}', '-o', 'tsv'],
        capture_output=True, text=True)

    for blob_name in result.stdout.strip().split('\n'):
        # Extraire la date du nom (db_YYYYMMDD_HHMMSS.bacpac)
        blob_date = datetime.strptime(...)
        if (datetime.now() - blob_date).days > retention_days:
            subprocess.run(['az', 'storage', 'blob', 'delete',
                '--account-name', config['storage_account'],
                '--container-name', 'raw',
                '--name', blob_name])
```

Listing 17 – Nettoyage automatique des anciens backups

A.6 — Restauration via Azure CLI

```
# Methode 1 : Restauration Point-in-Time (PITR)
# Restaure la base a un instant precis (dans les 14 derniers jours)
az sql db restore \
    --resource-group rg-projet-data \
    --server sqlelbrek-prod \
    --name projet_data_eng \
```

```
    --dest-name projet_data_eng_restored \
    --time "2026-02-16T10:00:00Z"

# Methode 2 : Import d'un fichier BACPAC depuis le Data Lake
# Recree la base a partir d'un export stocke dans le Data Lake
az sql db import \
    --resource-group rg-projet-data \
    --server sqlelbrek-prod \
    --name projet_data_eng_restored \
    --storage-key <storage_key> \
    --storage-key-type StorageAccessKey \
    --storage-uri "https://dlelbrek.blob.core.windows.net/raw/\
backups/projet_data_eng_20260217_143000.bacpac" \
    --admin-user sqladmin --admin-password ***
```

Listing 18 – Restauration via Azure CLI

A.7 — Tables du schema security (011_security_rls.sql)

```
-- 101 agences : villes > 10 000 hab dans la region
CREATE TABLE security.agences (
    agence_id        INT           IDENTITY(1,1) PRIMARY KEY,
    commune_code     NVARCHAR(10)  NOT NULL,
    ville            NVARCHAR(100) NOT NULL,
    departement_code NVARCHAR(3)   NOT NULL,
    taille_agence    NVARCHAR(20)  NOT NULL  -- 'GRANDE', 'MOYENNE', '
        PETITE'
);

-- Employes avec hierarchie (519 au total)
CREATE TABLE security.employes (
    employe_id          INT           IDENTITY(1,1) PRIMARY KEY,
    login_sql           NVARCHAR(100) NOT NULL UNIQUE,
    niveau_hierarchique NVARCHAR(30)  NOT NULL,
    agence_id           INT           NULL REFERENCES security.agences,
    departement_code    NVARCHAR(3)   NULL,   -- NULL = region entiere
    manager_id          INT           NULL REFERENCES security.employes
);

-- Mapping login -> departements autorises (alimente par ETL)
CREATE TABLE security.utilisateurs_zones (
    login_sql        NVARCHAR(100) NOT NULL,
    departement_code NVARCHAR(3)   NULL  -- NULL = acces region entiere
);
```

Listing 19 – Tables du schema security (011_security_rls.sql)

A.8 — Fonction predicat RLS (011_security_rls.sql)

```
CREATE FUNCTION security.fn_rls_geographie(@departement_code NVARCHAR
    (3))
RETURNS TABLE WITH SCHEMABINDING AS
RETURN
    SELECT 1 AS acces_autorise
    WHERE
        -- Utilisateur absent de utilisateurs_zones -> acces total
```

```
        -- (role_admin, role_analyst, role_etl_process)
        NOT EXISTS (
            SELECT 1 FROM security.utilisateurs_zones
            WHERE login_sql = USER_NAME()
        )
        OR
        -- Utilisateur enregistre (role_consultant) -> filtrer par dept
        EXISTS (
            SELECT 1 FROM security.utilisateurs_zones
            WHERE login_sql = USER_NAME()
              AND (departement_code = @departement_code
                   OR departement_code IS NULL)  -- NULL = region
                       entiere
        );
GO

CREATE SECURITY POLICY security.policy_rls_geographie
    ADD FILTER PREDICATE
        security.fn_rls_geographie(departement_code)
    ON dwh.dim_geographie
    WITH (STATE = ON, SCHEMABINDING = ON);
```

Listing 20 – Fonction predicat RLS (011_security_rls.sql)

A.9 — Extrait de code 9

```
CREATE VIEW dm.vm_nouveau_datamart AS
SELECT ... FROM dwh.fait_xxx f
JOIN dwh.dim_xxx d ON f.xxx_id = d.xxx_id
GROUP BY ...;
```

A.10 — Implementation SCD Type 1 – Ecrasement

```
CREATE PROCEDURE dwh.sp_scd_type1_activite
    @code_naf        NVARCHAR(10),
    @nouveau_libelle NVARCHAR(200)
AS
BEGIN
    -- Journaliser l'ancienne valeur (pour audit)
    INSERT INTO dwh.log_etl (etape, table_cible, statut, message)
    SELECT 'SCD_TYPE1', 'dim_activite', 'INFO',
        'Ancien libelle: ' + libelle_section
    FROM dwh.dim_activite
    WHERE code_section_naf = @code_naf;

    -- Ecrasement de la valeur
    UPDATE dwh.dim_activite
    SET libelle_section = @nouveau_libelle,
        date_modification = GETDATE()
    WHERE code_section_naf = @code_naf;
END;
```

Listing 21 – Implementation SCD Type 1 – Ecrasement

A.11 — Procedure SCD Type 2 – Historisation

```sql
CREATE PROCEDURE dwh.sp_scd_type2_geographie
    @code_commune     NVARCHAR(10),
    @nouveau_nom      NVARCHAR(200),
    @nouveau_code     NVARCHAR(10) = NULL
AS
BEGIN
    DECLARE @ancienne_version INT
    DECLARE @ancien_id INT

    -- Recuperer la version courante
    SELECT @ancienne_version = version, @ancien_id = geo_id
    FROM dwh.dim_geographie
    WHERE code_commune = @code_commune AND est_actif = 1;

    -- Fermer l'ancien enregistrement
    UPDATE dwh.dim_geographie
    SET date_fin_validite = GETDATE(),
        est_actif = 0,
        date_modification = GETDATE()
    WHERE geo_id = @ancien_id;

    -- Inserer la nouvelle version
    INSERT INTO dwh.dim_geographie (
        code_departement, nom_departement, code_region,
        nom_region, niveau, code_commune, nom_commune,
        latitude, longitude,
        date_debut_validite, date_fin_validite,
        est_actif, version,
        date_creation, date_modification
    )
    SELECT
        code_departement, nom_departement, code_region,
        nom_region, niveau,
        ISNULL(@nouveau_code, code_commune),
        @nouveau_nom,
        latitude, longitude,
        GETDATE(), NULL,
        1, @ancienne_version + 1,
        GETDATE(), GETDATE()
    FROM dwh.dim_geographie
    WHERE geo_id = @ancien_id;

    -- Journaliser le changement
    INSERT INTO dwh.log_etl (etape, table_cible, statut, message)
    VALUES ('SCD_TYPE2', 'dim_geographie', 'SUCCES',
        'Nouvelle version (v' + CAST(@ancienne_version + 1 AS VARCHAR)
        + ') pour commune ' + @code_commune
        + ' -> ' + @nouveau_nom);
END;
```

Listing 22 – Procedure SCD Type 2 – Historisation

A.12 — Procedure SCD Type 3

```sql
CREATE PROCEDURE dwh.sp_scd_type3_demographie
```

```sql
    @code_pcs         NVARCHAR(10),
    @nouveau_libelle NVARCHAR(200)
AS
BEGIN
    -- Sauvegarder l'ancien libelle et mettre a jour
    UPDATE dwh.dim_demographie
    SET ancien_libelle_pcs = libelle_pcs,
        libelle_pcs = @nouveau_libelle,
        date_changement_pcs = GETDATE(),
        date_modification = GETDATE()
    WHERE code_pcs = @code_pcs
      AND libelle_pcs <> @nouveau_libelle;

    INSERT INTO dwh.log_etl (etape, table_cible, statut, message)
    VALUES ('SCD_TYPE3', 'dim_demographie', 'SUCCES',
        'PCS ' + @code_pcs + ' mis a jour avec Type 3');
END;
```

Listing 23 – Procedure SCD Type 3

A.3 Glossaire

Terme	Definition
CSV	Comma-Separated Values – Format de fichier texte pour les donnees tabulaires
CDC	Change Data Capture – Mecanisme de capture des changements dans une base de donnees
DWH	Data Warehouse – Entrepot de donnees
ETL	Extract, Transform, Load – Processus d'extraction, transformation et chargement de donnees
ITIL	Information Technology Infrastructure Library – Referentiel de bonnes pratiques IT
KPI	Key Performance Indicator – Indicateur cle de performance
MCO	Maintien en Conditions Operationnelles
MERGE	Instruction SQL combinant INSERT et UPDATE en une seule operation
RBAC	Role-Based Access Control – Controle d'acces base sur les roles
RLS	Row-Level Security – Mecanisme SQL Server filtrant les lignes selon l'identite de l'utilisateur, de maniere transparente et non contournable
RGPD	Reglement General sur la Protection des Donnees
RPO	Recovery Point Objective – Perte de donnees maximale acceptable
RTO	Recovery Time Objective – Temps de restauration maximal acceptable
SCD	Slowly Changing Dimension – Dimension a evolution lente

TABLE 7 – Glossaire des termes techniques

A.4 Arborescence du projet

```
Projet-Data-ENG-E6/
  Terraform/
    main.tf                     # Infrastructure Azure + retention
       backups [E6]
```

```
  adf_linked_services.tf     # Data Factory
  rbac_configuration.tf      # RBAC Azure
  variables.tf / outputs.tf
  sql/
    001_create_schemas.sql   # 4 schemas (stg, dwh, dm, analytics)
    002_create_dimensions.sql # 6 dimensions
    003_create_facts.sql     # 7 tables de faits
    004_populate_dimensions.sql
    005_create_datamarts.sql # 5 datamarts + dashboard
    006_configure_security.sql # [E6] 4 roles RBAC (refonte + suppr.
       anciens roles)
    007_configure_performance.sql
    008_configure_logging.sql  # [E6] log_etl, log_erreurs, vues
       monitoring
    009_configure_backup.sql   # [E6] Vues monitoring backups
    010_scd_dimensions.sql     # [E6] SCD Type 1/2/3, MERGE, vues SCD
    011_security_rls.sql       # [E6] Schema security, RLS, agences,
       employes
    deploy_dwh.py
analytics/
  lib/
    data_prep.py             # [E6] Fonctions partagees ETL (
       load_communes, etc.)
  etl/
    run_etl.py               # Orchestrateur ETL [E6: logging +
       backup]
    load_dimensions.py       # Dimensions + SCD Type 1/2/3 [E6]
    load_facts.py            # Faits + emploi + menages [E6]
    backup_to_datalake.py    # [E6] Export BACPAC vers ADLS Gen2
    load_security.py         # [E6] Chargement agences, employes,
       zones RLS
    etl_notifier.py          # [E6] Notifications email (succes +
       erreur)
  tests/
    test_e6_evolution.py     # 10 tests E6 (journalisation, backup,
       sources, SCD, RBAC)
docs/
  E5_DATA_WAREHOUSE_GUIDE.md
  E6_MAINTENANCE_METHODOLOGY.md  # [E6] ITIL, incidents, KPIs
  E6_SCALABILITY_PROCEDURES.md   # [E6] Procedures d'evolution
rapports/
  E6_Evolution_Entrepot/
    rapport_E6.tex           # Ce rapport
uploads/
  landing/csv/
    EMPLOI_CHOMAGE_hauts_de_france.csv  # Source emploi [E6]
    Menage_hauts_de_france.csv          # Source menages [E6]
```