Lineage et usage d’un entrepôt PostgreSQL

Deux outils complémentaires : un explorateur de lineage construit par analyse statique du catalogue système, et un moniteur d’usage qui répond à « lesquels de ces milliers d’objets sont réellement utilisés ».

Fiches de méthode associées : `../../docs/09-data-catalog-lineage.md` et `../../docs/10-usage-des-donnees-nettoyage.md`

Contenu

Dossier	Rôle
`lineage-explorer/`	extraction du graphe de dépendances, lineage colonne, analyse d’impact, interface de navigation
`usage-monitor/`	collecte périodique des statistiques d’usage par type d’objet

lineage-explorer

La chaîne de traitement

```
pg_catalog  ──▸  extractor.py     requêtes sur le catalogue système, par lots
                      │
                      ▾
                 parser.py        analyse mixte : SQL déclaratif et procédural
                      │
                      ▾
                 graph.py         graphe orienté, 5 types d'arêtes
                      │
                      ▾
            column_lineage.py     propagation transitive colonne à colonne
                      │
                      ▾
              persistence.py      sérialisation binaire, écriture atomique
                      │
                      ▾
                   app/           interface de navigation et d'analyse d'impact
```

Ce qui vaut d’être regardé

1. L’extraction par lots avec repli dichotomique

(`lineage/extractor.py`)

La fonction du moteur qui rend la définition d’une vue est coûteuse : sur un catalogue de plusieurs milliers d’objets, un appel unitaire met des minutes. Le code extrait par lots, et sur échec d’un lot, le redécoupe récursivement en deux jusqu’à isoler l’objet fautif, qu’il abandonne seul.

Résultat : ni la lenteur d’un appel par objet, ni l’échec total du traitement sur un seul objet corrompu ou en dépassement de délai.

2. L’analyse mixte du SQL

(`lineage/parser.py`)

Un analyseur SQL traite parfaitement une vue déclarative mais ne comprend pas le langage procédural. L’approche retenue, pragmatique et documentée :

- expressions régulières sur le corps de la fonction pour repérer lectures et écritures ;
- analyseur SQL uniquement sur les blocs de requête extraits.

3. Les cinq types d’arêtes

(`lineage/graph.py`)

Vocabulaire minimal et suffisant pour décrire un entrepôt : table distante vers objet source, vue vers objets lus, fonction lit, fonction écrit, flux de données déduit.

Avec un filtrage explicite du bruit : les écritures vers les tables techniques (journalisation, suivi) polluent le graphe métier et sont exclues.

4. Le lineage colonne avec plafond de profondeur

(`lineage/column_lineage.py`)

Propagation transitive bornée pour survivre aux cycles. Résolution des alias vers le nom qualifié complet via le contexte de la vue.

5. Les limites documentées explicitement

(`CLAUDE.md`)

Sélection par étoile non résolue, SQL dynamique invisible, triggers non analysés, noms ambigus résolus au premier candidat.

Documenter le périmètre de non-couverture d'un outil d'analyse statique vaut mieux que laisser l'utilisateur le découvrir. Passées sous silence, ces limites deviennent des bugs de confiance.

6. La persistance atomique

(`lineage/persistence.py`)

Écriture dans un fichier temporaire puis renommage : un crash pendant l'écriture laisse le fichier précédent intact au lieu de produire un fichier tronqué lu par l'interface.

Sérialisation binaire compacte plutôt que JSON, plus un petit fichier de métadonnées séparé pour que l'interface affiche l'état sans charger tout le graphe.

7. Le masquage des identifiants

(`app/shared.py`)

Une trace du pilote de base contient l'hôte et parfois les identifiants complets. Une fonction de masquage par expression régulière sur le format d'URL de connexion est appliquée avant tout affichage, ce qui couvre les cas structurés même avec un hôte inconnu à l'avance.

8. L'analyse d'impact

(`lineage/analysis.py`)

Fonction générique sur tout graphe orienté : descendants d'un nœud, distance du plus court chemin, type de chaque objet impacté. Utile bien au-delà du nettoyage de patrimoine.

9. Le rafraîchissement en tâche de fond

(`app/`)

Verrou non bloquant pour empêcher deux traitements simultanés déclenchés par deux clics, avec état de progression partagé consulté à chaque rendu.

---

usage-monitor

Collecte périodique des statistiques d'usage, avec une abstraction qui encapsule l'hétérogénéité des sources.

Type d'objet	Source	Disponibilité
table, vue matérialisée, table distante	compteurs d'accès natifs	immédiate
fonction, procédure	compteurs d'appels	nécessite une activation en configuration
vue ordinaire	aucune statistique native	repli peu fiable

Le point le plus instructif : le repli pour les vues ordinaires a été tenté puis abandonné après mesure réelle (couverture nulle constatée). Vérifier empiriquement une hypothèse de mesure avant de bâtir une fonctionnalité dessus, plutôt que la maintenir avec une fiabilité inconnue.

La couche de présentation ne voit qu'un état homogène à trois valeurs (utilisé, incertain, inutilisé). Le troisième état est ce qui rend l'analyse honnête : sans lui, on classe « inutilisé » ce qu'on n'a pas su mesurer, et un nettoyage sur cette base casse la production.

Vérification après anonymisation

Contrôle	Résultat
Tests (`lineage-explorer/tests/`)	112 passent
Compilation des 24 fichiers Python	ok
Imports rendus autonomes	37 corrigés
Fichiers d'exemple de configuration	placeholders uniquement
Motifs sensibles résiduels	aucun

Note : aucun identifiant métier n'était codé en dur dans ce projet, tout passe par la configuration. C'est en soi un bon signe de conception.

Ce qui a été retiré

Retiré	Raison
`data/graph.msgpack`, `data/graph_meta.json`	graphe réel de production : noms d'objets et structure interne
`data/lineage_*.json`	export de lineage réel
`src/test.ipynb`	notebook pouvant contenir des sorties de données
`.env`	secrets réels
`.claude/`	permissions locales personnelles
historique de contrôle de version	contient ce qui a été supprimé

Le fichier d'exemple de `lineage-explorer` contenait un nom d'hôte interne réel : les valeurs ont été remplacées par des marqueurs, en conservant les valeurs neutres (ports, couleurs, seuils) qui documentent les formats attendus.

Imports : le projet vivait dans une arborescence plus large ; les imports ont été rendus relatifs au projet pour qu'il soit autonome.