TEKELI Emin-Can
BARILE Nicolas
KARABAY Mehmet
BOUHARRA Mehdi

Rapport Conception d'un Datawarehouse - Reporting & Dashboarding

Introduction

Dans le cadre de notre projet de "Conception d'un Datawarehouse - Reporting & Dashboarding", nous avons dû concevoir un entrepôt de données qui répond aux besoins d'une compagnie d'assurance. Dans ce rapport, nous présenterons dans un premier temps les outils utilisés, le jeu de données et la problématique. Ensuite, nous expliquerons la modélisation et la réalisation du datawarehouse en détail. Enfin,nous conclurons.

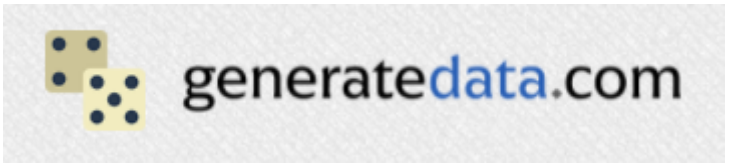
Outils utilisés

Le datawarehouse a été hébergé à l'aide de PostgreSQL. Les jobs ETL sont quant à eux réalisés sur Talend Open Studio.

Nous avons tous utilisé le système d'exploitation Windows 10/11.

Jeu de données

Les données ont été générées sur https://generatedata.com/.



Problématique

Une compagnie d'assurance possède une application transactionnelle qui permet de gérer les polices (contrats) de ses clients ainsi que les sinistres (accidents) déclarés par ces clients. Elle s'occupe de trois types de risques: automobile, immobilier, responsabilité civile.

Dans le système opérationnel, l'employé ou l'agent peut :

- mettre à jour ou supprimer une police d'assurance
- mettre à jour un risque pour une police donnée
- mettre à jour ou supprimer une voiture/maison
- chiffrer la police.

Concernant les transactions, on a un grand nombre d'infos comme par exemple:

- date de création (du contrat)
- date d'effet (début d'assurance)
- client
- agent
- risque
- police
- l'état.

Pour gérer les sinistres déclarés par les clients, les employés ou agents d'assurances ont à leurs disposition des outils touchant:

- à la déclaration de sinistres
- à la gestion d'expertise
- à la gestion des paiements
- au client
- aux biens sinistrés
- à la police
- aux montants financiers payés ou qui restent à payer.

On s'intéresse uniquement à la globalisation des transactions par mois. Pour chaque bien assuré, on veut le montant de la prime. On veut suivre l'état de la police en spécifiant un contrat nouvellement créé, modifié, en cours ou juste clôt. On veut sortir des tableaux par client, agent ou employé, état des faits, avec toutes les sommations possibles (montants), par sinistre, avec le total payé par mois, par bien...

Modélisation

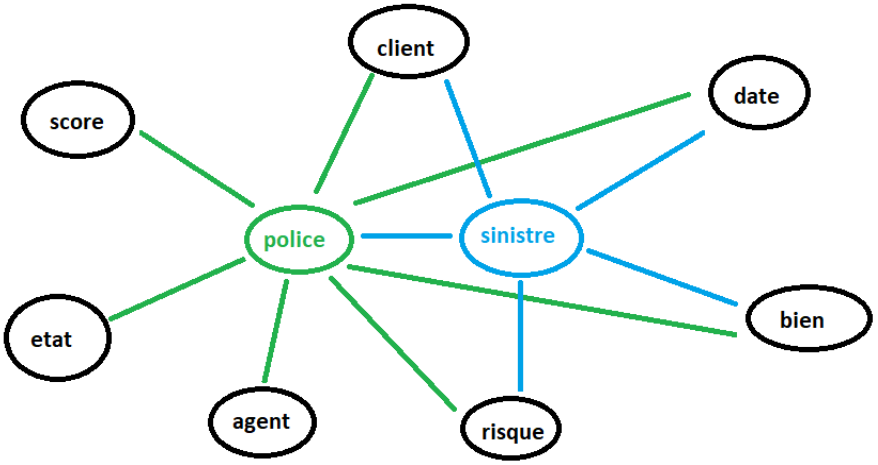
Notre solution comporte 7 dimensions:

- Agent
- Client
- Bien (bien assuré)
- Risque (type de bien: A pour automobile, H pour habitation, RC pour responsabilité civile)
- Score (score de 1 à 5 pour évaluer la police)
- Etat (4 états possibles: ENCOURS, CREE, MODIFIE, CLOS)
- Date.

Elle comporte également 2 faits:

- Police
- Sinistre.

Le schéma relationnel est le suivant:



Extraction, transformation et chargement des données

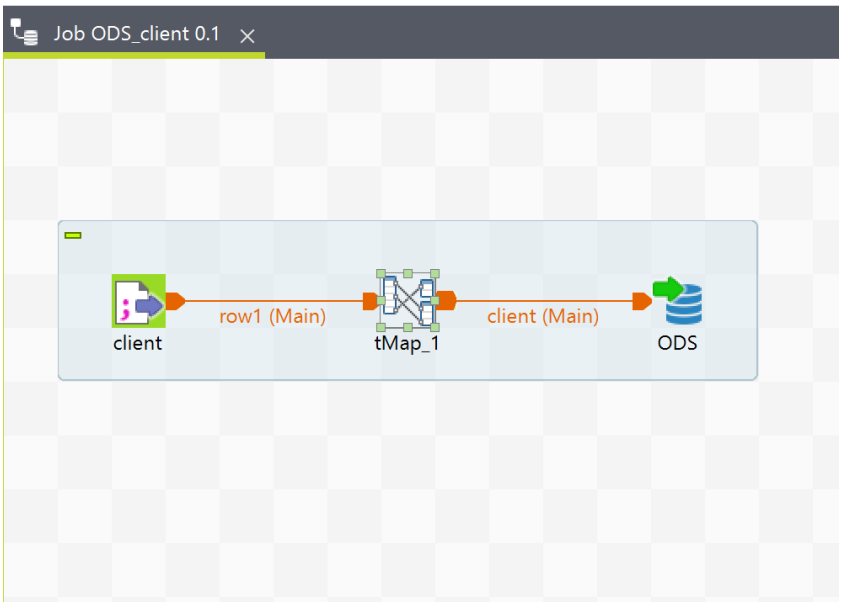
Avant d'atterrir dans l'entrepôt de données, les données sont dans un premier temps stockées dans une zone tampon: l'ODS. Aucune transformation n'est appliquée dans cette partie, les données sont simplement stockées dans cette zone. Nous appliquons ensuite un certain nombre de transformations et de vérifications sur les données avant de les stocker dans l'entrepôt.

Nous allons maintenant voir en détail les parcours types des données. Elles sont traitées différemment si elles représentent une dimension ou un fait.

Jobs types traitant une dimension

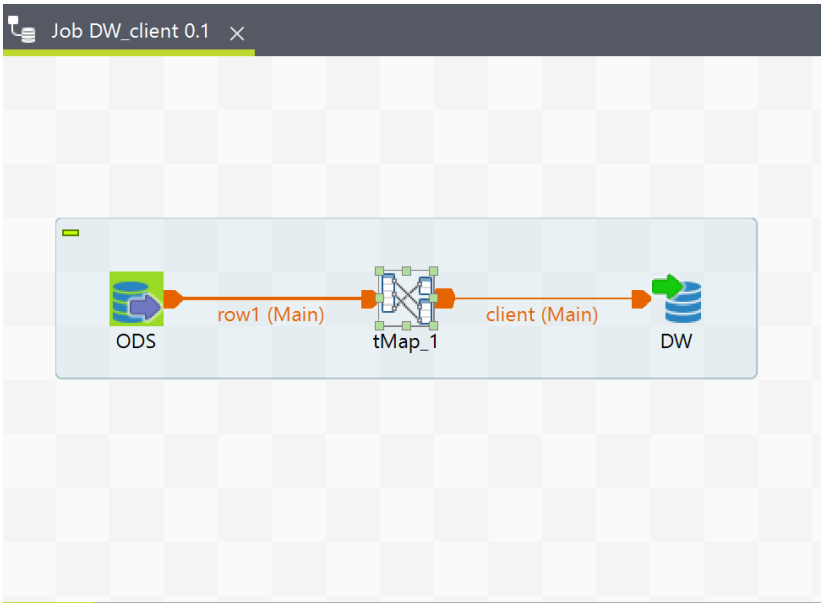
CSV vers ODS

Stockage dans l'ODS depuis le fichier CSV. Un simple chargement: on lit les données depuis le csv à l'aide d'un tFileInputDelimited, on dirige ensuite les données à l'aide d'un tMap, enfin nous les stockons dans l'ODS (tDBOutput).



ODS vers entrepôt

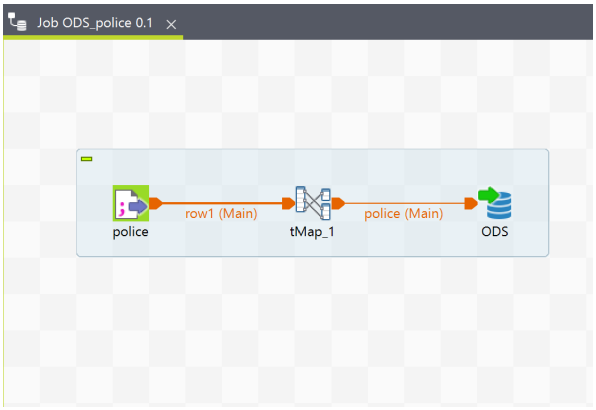
Stockage dans l'entrepôt depuis l'ODS. Aucun changement à effectuer. Un simple chargement: on lit les données depuis l'ODS à l'aide d'un tDBIput, on dirige ensuite les données à l'aide d'un tMap, enfin nous les stockons dans l'entrepôt de données (tDBOutput).



Jobs types traitant un fait

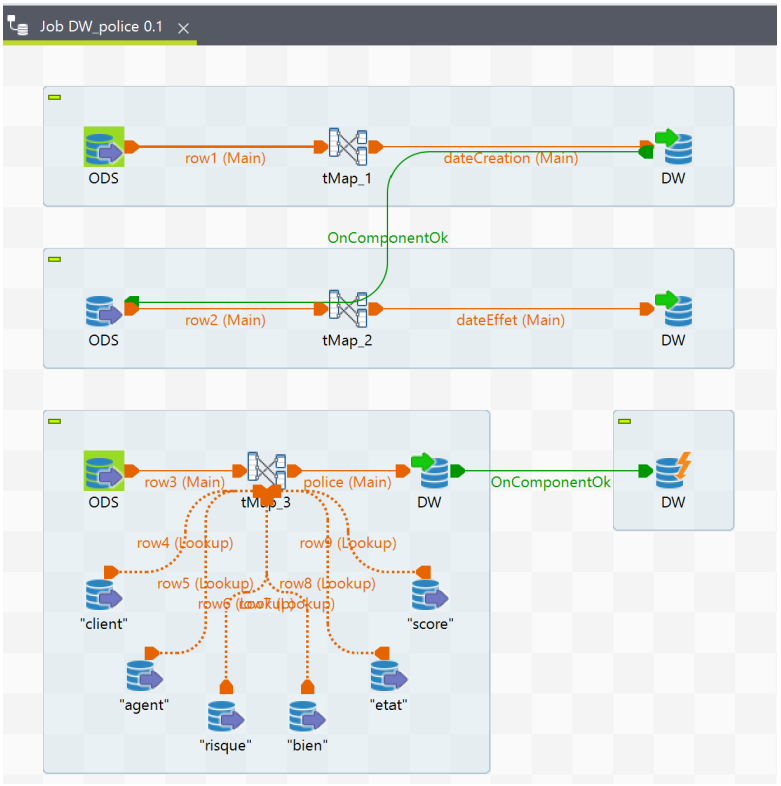
CSV vers ODS

Identique à une dimension, rien ne change.

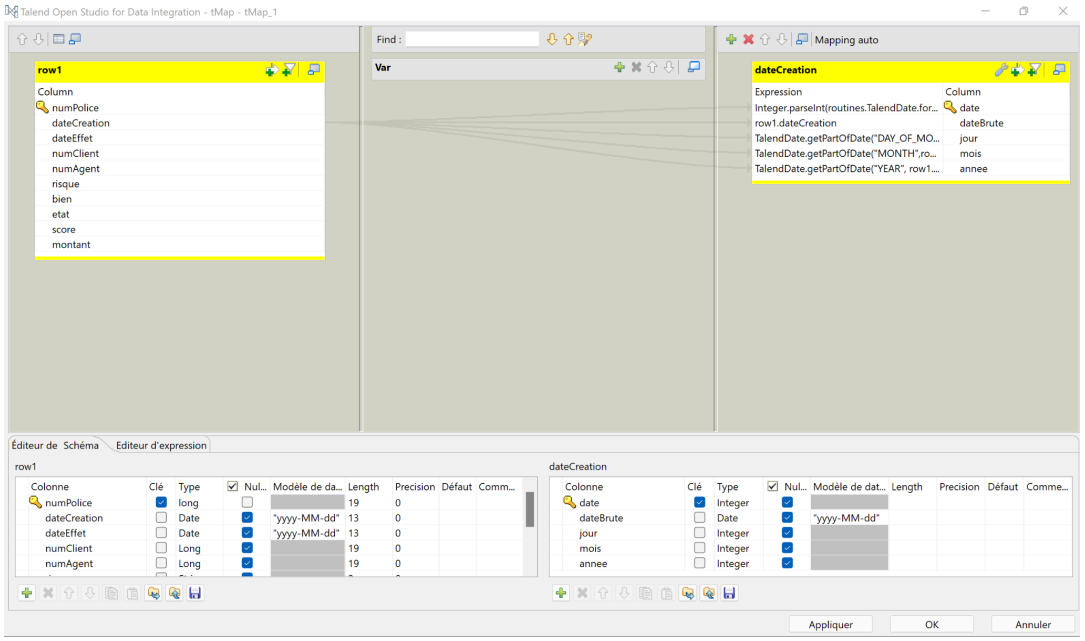


ODS vers entrepôt

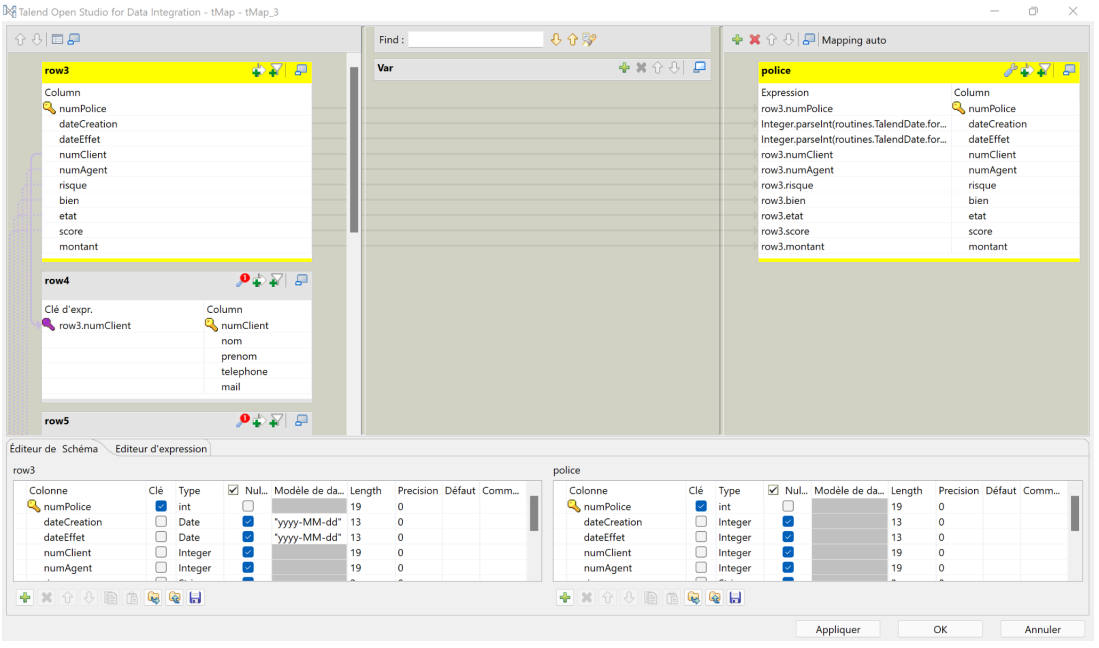
Ici, on effectue une transformation au niveau des dates. On effectue également un contrôle d'intégrité avec les tables de dimensions en look-up pour éviter d'avoir des clés étrangères inconnues. Enfin, on assigne les clés étrangères aux champs concernés dans les tables de faits à l'aide d'un tDBRow (script SQL).



Transformation de la date. La clé principale devient un entier obtenu avec le format YYYYMMDD (par exemple, 31/01/2022 devient 20220131). On a à l'aide de cette méthode un entier unique qui prend peu de place en mémoire que l'on peut utiliser comme clé principale. On peut ensuite ajouter à la dimension Date la date concernée avec plus de détails. L'avantage est que seules les dates dont on a besoin sont créées.



Stockage du fait dans l'entrepôt. On s'assure de l'intégrité des données à l'aide des tables en look-up.



Assignation des clés étrangères à l'aide d'un script SQL. On s'assure dans un premier temps de supprimer les contraintes si elles existent déjà. On peut ensuite faire notre assignation.

DW(DW).sql

DW Mode de contexte

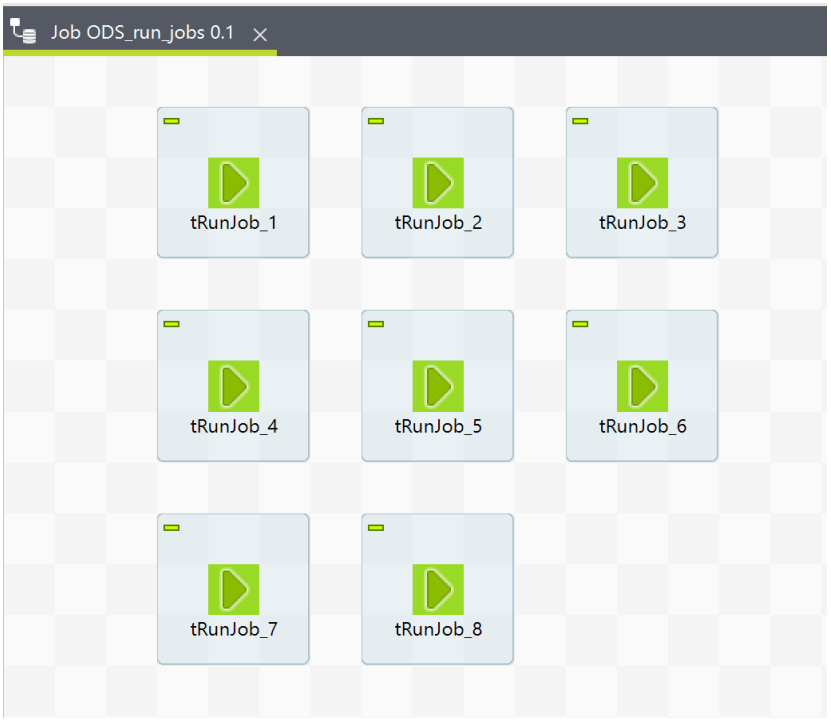
```
ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_dateCreation_fkey";
ALTER TABLE "police" ADD FOREIGN KEY ("dateCreation") REFERENCES date("date");

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_dateEffet_fkey";
ALTER TABLE "police" ADD FOREIGN KEY ("dateEffet") REFERENCES date(date);

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_numClient_fkey";
ALTER TABLE "police" ADD FOREIGN KEY ("numClient") REFERENCES client("numClient");

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_numAgent_fkey";
ALTER TABLE "police" ADD FOREIGN KEY ("numAgent") REFERENCES agent("numAgent");

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_risque_fkey";
ALTER TABLE "police" ADD FOREIGN KEY (risque) REFERENCES risque(risque);

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_bien_fkey";
ALTER TABLE "police" ADD FOREIGN KEY (bien) REFERENCES bien(bien);

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_etat_fkey";
ALTER TABLE "police" ADD FOREIGN KEY (etat) REFERENCES etat(etat);

ALTER TABLE "police" DROP CONSTRAINT IF EXISTS  "police_score_fkey";
```

Automatisation du lancement des jobs

Nous avons créé deux jobs automatisant les jobs concernant l'ODS (staging) et les jobs concernant l'entrepôt de données. Certains jobs n'ont pas besoin d'être lancés régulièrement, comme celui qui traite la table Etat, car ces tables varient rarement. On peut donc traiter ces tables une seule fois à l'initialisation ou chaque fois qu'on effectue une mise à jour. Elles n'ont par ailleurs pas nécessairement besoin de passer par l'ODS. Par soucis d'universalité, nous avons pris la décision de faire passer ces jobs par l'ODS et de les exécuter à chaque fois. Cela nous permet de ne pas avoir à faire au cas par cas. Le temps nécessaire à l'exécution de ces jobs est par ailleurs négligeable par rapport aux autres jobs donc nous n'aurons pas de soucis de complexité.

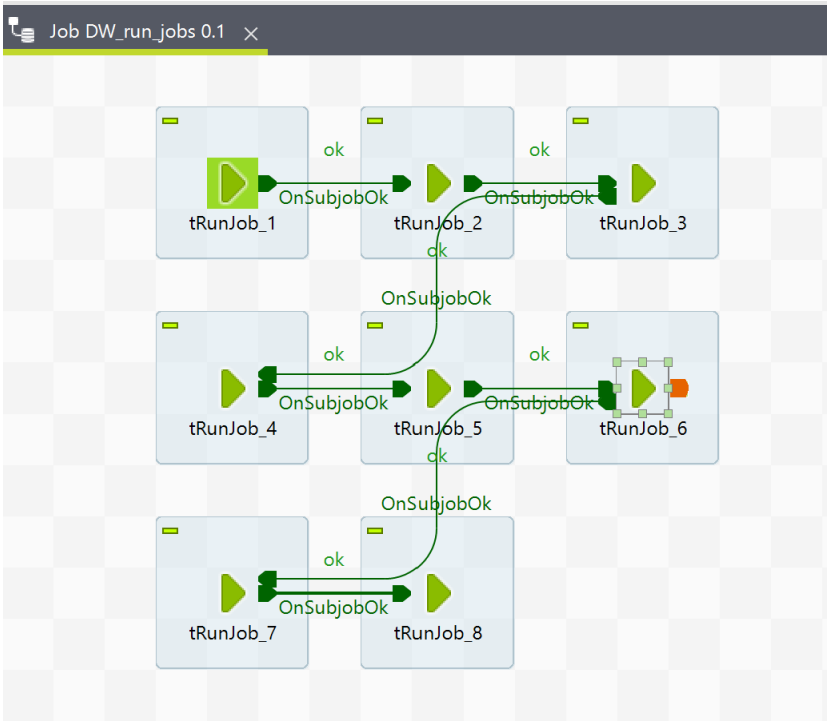
Jobs ODS

Simple lancement de tous les jobs en parallèle.



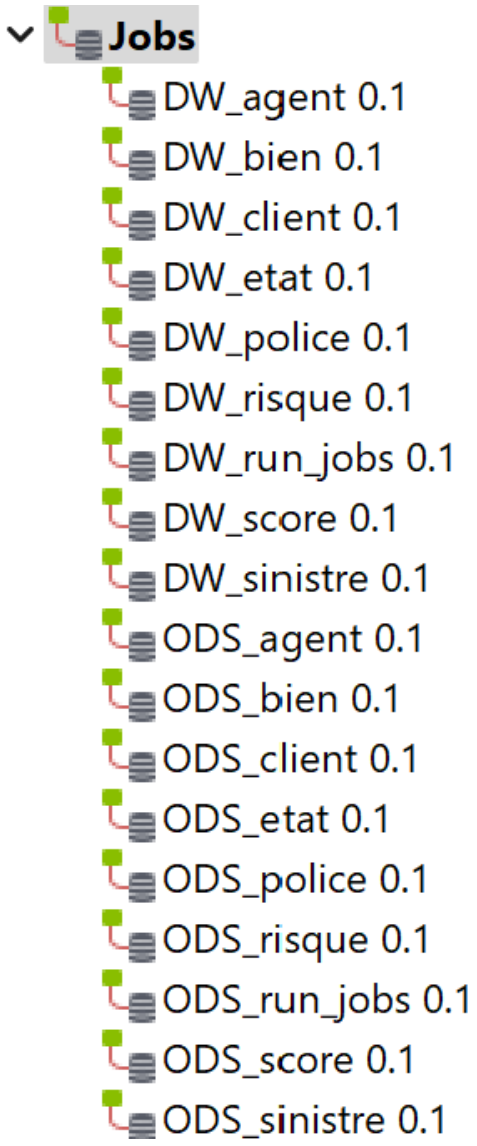
Jobs DW

On doit s'assurer de lancer les jobs dans le bon ordre car les faits possèdent des clés étrangères. On doit donc stocker les tables avec les clés dépendantes avant pour ne pas avoir de problème pendant le contrôle d'intégrité.

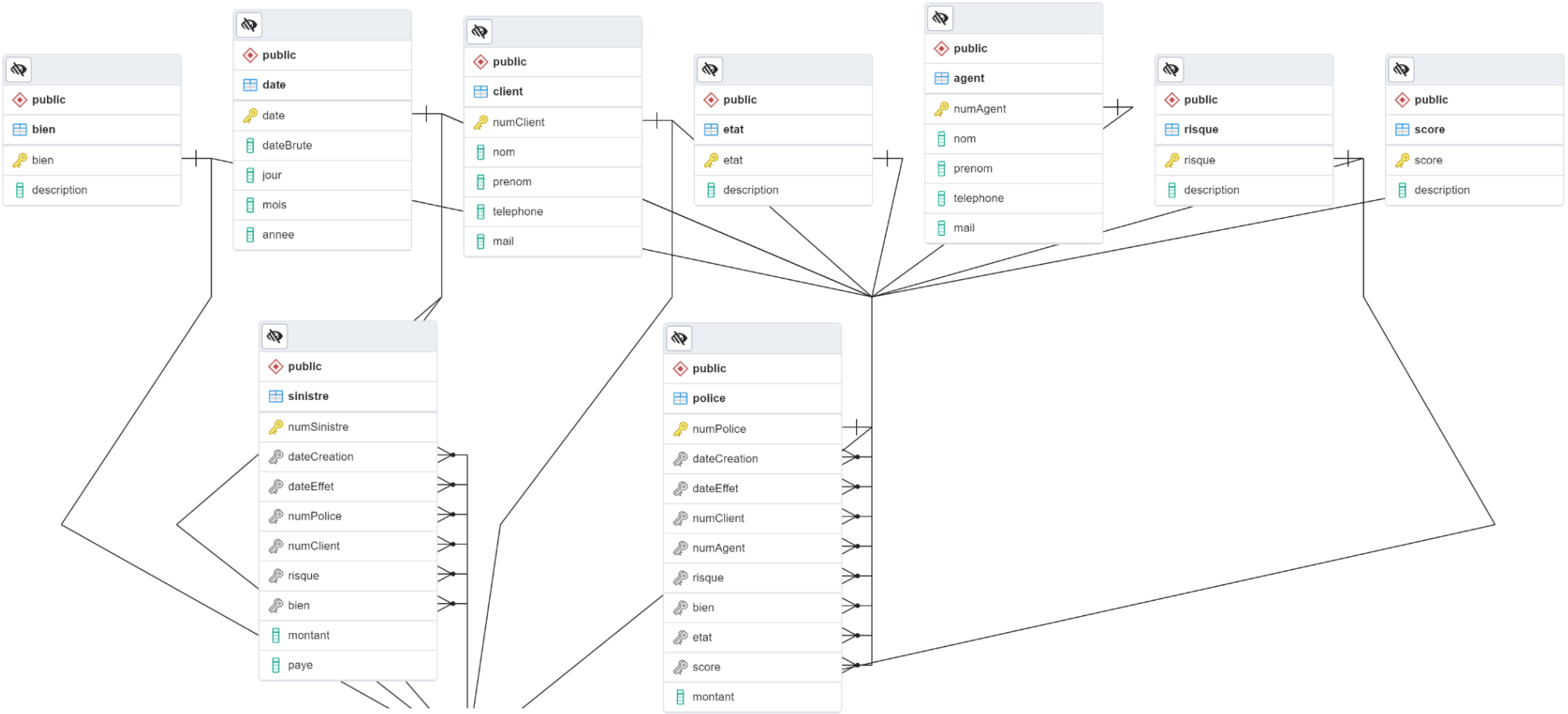


Liste et hiérarchie des jobs

On a décidé d'avoir un job par table et par destination (ODS ou DW). Cela nous permet d'avoir des jobs simples et le travail collaboratif est facilité. Le désavantage est le grand nombre de jobs.



Modélisation complète



Conclusion

Nous avons dans ce projet appris à concevoir et à réaliser un entrepôt de données pour un problème d'assurance à l'aide d'outils comme Talend ou PostgreSQL.

L'entrepôt conçu reste très basique et loin d'un réel entrepôt de données mais l'important concernant la conception et la réalisation d'un entrepôt de données avec des dimensions et des faits a été abordé durant le projet.