



Projet de migration de Talend vers Data Build Tool 2024-2025

Chloé LEYRIS

chloe.leyris@etu.univ-grenoble-alpes.fr

Université Grenoble Alpes – 3ème année de BUT science des données

Recommerce Group

Tuteur académique : BRAULT Vincent

Tuteur entreprise : ARTIGUE Thomas

Remerciement

Je tiens tout d'abord à exprimer ma gratitude à Thomas Artigue, mon tuteur de stage, pour son soutien précieux tout au long de mon alternance. Dès le début, il a pris le temps de m'expliquer en détail la structure de l'entreprise, son fonctionnement, ainsi que les différents projets et les tâches qui allaient me concerner. Grâce à ses explications claires et précises, ainsi qu'à sa grande patience, j'ai compris les enjeux de mon rôle au sein de l'équipe et j'ai pu appréhender mon alternance sereinement. Il a toujours été présent, offrant son expertise et sa bienveillance, ce que j'ai grandement apprécié.

Je remercie également toute l'équipe DIP pour leur accueil chaleureux et leur soutien continu durant cette année. L'ambiance conviviale au bureau a grandement contribué à mon bien-être professionnel, et je suis reconnaissante pour la confiance qu'ils m'ont accordée et pour l'opportunité de travailler avec eux.

Enfin, je tiens à exprimer ma reconnaissance envers mon tuteur académique, Vincent Brault, pour ses remarques, ses conseils constructifs et d'avoir ouvert le dialogue. Sa contribution m'a permis d'améliorer la qualité de mon rapport et d'identifier des axes d'amélioration essentiels pour ma soutenance finale. Grâce à ses recommandations, j'ai pu apporter les ajustements nécessaires et me préparer de manière optimale pour présenter mon travail.

Résumé

Dans le cadre de l'évolution des outils de traitement de données, l'entreprise Recommerce a entrepris un projet de migration de sa plateforme d'intégration de données depuis Talend vers Data Build Tool (DBT). Ce changement a pour objectif de moderniser l'infrastructure analytique, de faciliter la maintenance, d'optimiser les performances et de standardiser les pratiques de développement. Le projet s'est articulé autour de plusieurs axes : la modélisation des données avec DBT, la réorganisation des schémas en fonction des domaines métiers, la mise en place de procédures stockées en Snowflake, et la rédaction de tests automatisés. Une attention particulière a été portée à la documentation et à la structuration du code pour assurer la pérennité du projet. À ce jour, des modèles fonctionnels ont été déployés, accompagnés de pipelines stables et documentés. Une organisation claire des rôles au sein de l'équipe a permis une collaboration fluide entre les profils techniques et métiers. Le projet est conduit dans le respect d'un calendrier serré, avec comme échéance majeure la fin de la licence Talend prévue au 15 novembre.

Mots-clés : migration de données, DBT, Talend, Snowflake, modélisation, pipelines, tests automatisés, infrastructure analytique.

Abstract

Title: Migration from Talend to Data Build Tool: Data Modeling and Workflow Modernization in a Snowflake Environment

This project presents the migration of Recommerce's data integration workflows from the Talend platform to Data Build Tool (DBT). The primary objective was to modernize the analytical infrastructure, improve maintainability, and enforce standardized development practices. The migration process included data modeling in DBT, implementation of pre- and post-execution procedures using Snowflake stored procedures, definition of clear modeling guidelines, and the integration of automated testing. Team members were assigned functional schemas based on domain knowledge to ensure both technical consistency and business alignment. DBT models were successfully developed and deployed, resulting in a robust and maintainable data pipeline. Extensive documentation was produced to support knowledge transfer and scalability. The project follows a tight schedule due to the upcoming deactivation of the Talend license on November 15, making coordination and workflow automation critical success factors.

Keywords: data migration, DBT, Talend, Snowflake, data modeling, automated testing, workflow orchestration.

Table des matières

I.	Introduction	5
II.	Présentation de l'entreprise	5
A.	Historique de Recommerce	6
B.	Le principe de recommercialisation.....	7
C.	La Data Intelligence and Pricing (DIP)	7
III.	Mise en contexte	9
A.	Limite de talend	9
B.	Raisons du choix de DBT	9
C.	Le processus de prise de décision.....	10
IV.	Les outils utiliser	10
A.	Fonctionnement de dbt.....	10
B.	Organisation des fichiers dans dbt	11
V.	Mise en place du projet	12
A.	Organisation	12
B.	Création des models	12
C.	Construction de la table ODS_BUYBACK.....	14
1.	CTE.....	14
2.	Structure de la table	16
3.	Configuration.....	17
VI.	L'évolution des model au cours de l'alternance	18
A.	Incrementation	18
1.	Mise en place du merge	19
2.	Quelle que exception.....	19
3.	Macros	20
B.	Gestion des sources.....	20
C.	Model python	21
D.	Intégration des tests de qualité des données.....	21
E.	Procedure	21
VII.	Suite du projet avec la base de donner Recommerce	22
1.	guidelines.....	22
2.	Les source	23
3.	Repartition des rôles.....	23
VIII.	Conclusion	24
IX.	Glossair	25
X.	Webographie	26
XI.	Annexes	26

I. Introduction

Dans la continuité de mon stage de deuxième année de BUT Science des données, j'effectue actuellement mon alternance de troisième année au sein de l'entreprise Recommerce Solutions, dans l'équipe DIP (Data Intelligence Pricing). Lors de mon stage, j'avais mené une étude de faisabilité sur la migration de l'ensemble de la base de données de Recommerce de Talend vers DBT. Mon rôle cette année est de concrétiser cette migration, avec l'aide de l'équipe data.

L'objectif principal de mon alternance est donc la mise en œuvre effective de cette migration, en remplaçant l'outil Talend par DBT, dans une logique de modernisation et d'optimisation des pipelines de données de l'entreprise. J'ai notamment été chargée de la migration des traitements liés à la base de données de CircularX, ancienne filiale récemment intégrée à Recommerce. Ce travail a nécessité une alternance entre des phases d'analyse (compréhension des données, étude de l'existant, choix techniques) et des phases de développement, au cours desquelles j'ai conçu et implémenté des modèles DBT adaptés à l'environnement Snowflake et à ses spécificités.

Dans ce rapport, je commencerai par présenter l'historique et l'organisation de l'entreprise afin de fournir les éléments nécessaires à la compréhension du contexte du projet. J'expliquerai ensuite la mise en place progressive de la migration, ce qui permettra d'illustrer les différentes étapes techniques et organisationnelles franchies. Cette partie sera suivie d'une analyse des améliorations apportées tout au long de l'année, reflet de l'évolution continue du projet. Enfin, je développerai les actions mises en place pour assurer un transfert de connaissances durable au sein de l'équipe, avant de conclure sur les résultats obtenus et les perspectives envisagées pour la suite du projet durant la période estivale.

II. Présentation de l'entreprise

L'essor du marché des produits reconditionnés ne cesse de croître. Parallèlement, les enjeux écologiques actuels, couplés aux évolutions comportementales des consommateurs, renforcent la dynamique de l'économie circulaire. Selon une étude relayée par Le Journal du Dimanche en 2024, la part des Français utilisant un smartphone reconditionné est passée de 7 % en 2018 à 20 %, et 56 % des consommateurs se disent aujourd'hui intéressés par ce type d'achat. Cette tendance traduit une évolution durable des habitudes de consommation. Dans ce contexte porteur, des acteurs innovants comme Recommerce Group émergent et s'affirment sur ce marché en pleine expansion.

A. Historique de Recommerce

En 2009, animés par leur engagement pour la durabilité et les enjeux écologiques, Pierre-Étienne Roinat, Benoît Varin, Cédric Maucourt et Antoine Jeanjean fondent RECOMMERCE GROUP, initialement sous la forme du site MonExTel. Ce service proposait une approche innovante du recyclage de téléphones portables : l'utilisateur pouvait estimer la valeur de reprise de son appareil via un questionnaire simple, puis en faire don à une association de son choix. En moins de cinq minutes, il réalisait ainsi un geste à la fois écologique et solidaire. Les téléphones étaient ensuite remis en état ou recyclés, selon leur état, dans une logique de valorisation durable. Rachatdemobile.com était l'un de leur partenaire, c'est sur leur site que j'ai trouvé les informations.

Cette première initiative, centrée sur l'impact environnemental et sociétal, a jeté les bases de ce qui deviendra Recommerce. L'entreprise s'est progressivement structurée autour de la conviction que les produits issus du réemploi peuvent entamer un nouveau cycle commercial. À travers Recommerce, les fondateurs ont su intégrer les principes de l'économie circulaire (voir Figure 1) au secteur en constante évolution des télécommunications, ouvrant ainsi de nouvelles perspectives économiques tout en répondant à des enjeux environnementaux majeurs (Recommerce.com).

Pionniers sur le marché, leur aventure débute en France avant de s'étendre à l'échelle européenne. Aujourd'hui, leurs produits sont diffusés dans plus de 15 pays et l'entreprise est implantée physiquement dans 8 pays européens, notamment en Allemagne, en Suisse et en Espagne (Recommerce.com). RECOMMERCE GROUP emploie actuellement près de 200 collaborateurs.

L'entreprise est spécialisée dans l'achat, la remise à neuf et la revente de produits électroniques. Elle collabore avec de nombreux points de vente en marque blanche. Si elle s'est initialement concentrée sur les smartphones et appareils high-tech, elle prévoit désormais d'élargir son champ d'action aux appareils électroménagers et autres (Recommerce.com, 2024). Son approche repose sur l'utilisation de technologies et de méthodes avancées couvrant l'ensemble du processus : de la collecte des produits à leur revente. Cette stratégie vise à prolonger la durée de vie des équipements tout en réduisant l'impact environnemental de leur cycle de vie.

En tant qu'alternante au sein de l'entreprise, j'ai eu l'opportunité de participer aux séminaires et réunions de communication, ce qui m'a permis de mieux comprendre l'histoire de Recommerce, ses ambitions stratégiques, ainsi que les valeurs fortes qui animent ses équipes : innovation, durabilité, transparence et circularité. Ces moments d'échange m'ont donné une vision plus large de l'entreprise et de son positionnement sur le marché du reconditionné.

L'économie circulaire

3 domaines, 7 piliers



Figure 1: Schéma du principe d'économie circulaire Source : [ADEME – Agence de la transition écologique](#)

B. Le principe de recommercialisation

Comme le souligne Eco CO2, le recommerce connaît une croissance portée par les préoccupations environnementales et les évolutions des comportements d'achat numériques [Eco CO2, Le recommerce : une tendance qui se confirme]. Recommerce Group intervient dans la remise sur le marché des produits d'occasion, en agissant comme intermédiaire entre les détenteurs de ces produits et les distributeurs. Les produits peuvent ainsi être collectés, remis en état, puis revendus.

Cette démarche concerne aussi bien les consommateurs particuliers que les entreprises, qui peuvent y recourir dans le cadre de stratégies de gestion budgétaire ou pour répondre à des engagements environnementaux.

Afin d'adresser cette diversité de besoins, Recommerce Group propose des offres adaptées à plusieurs types de clientèle. Parmi ses partenaires commerciaux figurent notamment Amazon, Boulanger, Bouygues Telecom et d'autres enseignes, d'après les données internes observées dans le cadre de mon alternance.

C. La Data Intelligence and Pricing (DIP)

Je travaille au sein d'une équipe appelée DIP (Data Intelligence and Pricing), basée à Grenoble dans un hub dédié. Cette équipe est composée de 11 personnes aux profils complémentaires.

Elle accompagne l'entreprise dans l'optimisation des processus d'achat, de reconditionnement et de revente à travers l'exploitation des données.

La DIP est organisée autour de deux pôles principaux :

- Le pôle pricing, chargé de la fixation des prix des produits reconditionnés.
- Le pôle data, dédié à la manipulation et à l'analyse des données de l'entreprise.

Ce second pôle a pour mission d'assurer une intégration fiable et cohérente des données dans les systèmes, de répondre aux besoins des différents métiers, et de produire des analyses stratégiques. Ces analyses permettent de suivre la performance de l'entreprise, d'orienter les décisions et d'optimiser la marge bénéficiaire. Les deux sous-équipes fonctionnent de manière complémentaire, favorisant ainsi le partage des connaissances métiers et techniques.

Les principaux objectifs de l'équipe DIP sont les suivants :

- Contribuer à un monde plus durable en luttant contre le gaspillage et en promouvant une consommation responsable.
- Encourager la collecte des produits usagés et garantir la qualité de leur reconditionnement.
- Développer des stratégies de pricing qui soutiennent l'activité de recommerce.
- Fournir des données fiables pour maximiser la qualité des services et leur rentabilité.

L'organisation de l'équipe repose aussi sur un système de streams, c'est-à-dire de groupes pluridisciplinaires travaillant régulièrement ensemble sur des projets spécifiques. Parmi ces streams, on trouve notamment :

Le stream Market Intelligence, chargé de produire des informations stratégiques (prix, coûts, indices de marché) sur la concurrence.

Le stream Pricing Models, qui construit les modèles de tarification et veille à l'alignement des décisions avec les intérêts économiques de l'entreprise.

Le stream Référentiel, qui propose des réponses systématiques à des problématiques opérationnelles quotidiennes.

Pour assurer le bon déroulement de mon projet, je suis accompagnée par plusieurs collaborateurs clés : l'ensemble de l'équipe DIP dirigée par Matthieu Kraan ; mon maître d'apprentissage Thomas Artigue, qui encadre mon travail ; ainsi qu'Ahlem Gharsallah, ancienne alternante désormais en CDI. Je suis également en lien avec Augustin Poyet, responsable des données de CircularX, filiale de Recommerce.

III. Mise en conexte

Dans cette partie, j'explique pourquoi l'équipe Data Intelligence and Pricing a envisagé une transition de Talend vers dbt, et en quoi mon profil d'étudiante en science des donnée ou j'ai suivi les cours de Bases de données relationnelles 1 et 2, Systèmes d'information décisionnels et Big Data : enjeux, stockage et extraction notamment et ou j'ai participer a des situation d'apprentissage et d'évaluation telle que Reporting à partir de données stockées dans un SGBD relationnel, Conception et implémentation d'une base de données, Intégration de données dans un datawarehouse et Migration de données vers ou depuis un environnement NoSQL. Bien que d'autre ressource et SAE mon aussi aider durant cette année, celle si sont les principales et celle qui ont justifié ma participation à ce projet. Cette mise en contexte permet de comprendre les limites rencontrées, les raisons du choix de dbt, et le rôle que j'ai joué dans le processus de prise de décision.

A. Limite de talend

Dans un contexte de croissance rapide, les limites de Talend deviennent de plus en plus visibles. Bien que l'outil propose une interface intuitive basée sur le glisser-déposer, cette facilité d'utilisation peut masquer une complexité croissante à mesure que les flux de données se multiplient. Chaque nouveau besoin métier nécessite la création ou la duplication de composants, ce qui alourdit l'architecture globale des projets. Par ailleurs, le coût des licences, qui coute Recommerce 18000 euro par ans, représente une contrainte significative pour une entreprise en pleine expansion : plus l'équipe s'agrandit, plus les frais d'utilisation de Talend augmentent. À cela s'ajoutent les coûts de maintenance, notamment liés à la nécessité d'avoir des profils techniques qualifiés pour gérer les projets, résoudre les erreurs de compilation ou maintenir les serveurs d'exécution. Enfin, la rigidité de la plateforme rend parfois difficile l'intégration rapide de nouvelles technologies ou méthodologies de travail, comme l'utilisation systématique du versionnage via Git ou le déploiement continu. Ces limites freinent l'agilité de l'équipe data, pourtant cruciale pour accompagner efficacement la croissance de l'entreprise.

B. Raisons du choix de DBT

dbt Core (Data Build Tool) est un outil open source qui permet de transformer les données directement dans l'entrepôt de données (comme Snowflake, BigQuery ou Redshift) à l'aide de requêtes SQL organisées sous forme de modèles. Contrairement à Talend, dbt n'utilise pas d'interface graphique : il repose sur une approche code-first, ce qui facilite le versionnage via Git, la collaboration entre développeurs, et l'intégration dans des pipelines de déploiement continu (CI/CD). Les transformations sont regroupées en fichiers .sql et documentées à l'aide de fichiers .yml, ce qui favorise la lisibilité et la reproductibilité des traitements. Chaque modèle peut dépendre d'un autre, permettant à dbt de générer automatiquement un graphe de dépendances entre les tables. Cela rend la maintenance plus fluide et réduit le risque d'erreurs liées aux modifications non contrôlées. Par ailleurs, dbt étant gratuit dans sa version Core, il permet de réduire considérablement les coûts de licence tout en offrant une scalabilité plus adaptée à une entreprise en pleine croissance. Donc dbt reste efficace même pour gérer une augmentation du volume de données, d'utilisateurs ou de tâches sans perte de performance.

C. Le processus de prise de décision

Les conclusions concernant les avantages de dbt ont d'abord émergé durant mon stage de deuxième année, à travers une étude de faisabilité puis une première mise en œuvre concrète sur un flux stratégique. En rejoignant Recommerce pour un stage de trois mois, je n'avais encore jamais utilisé dbt, mais j'ai rapidement pu monter en compétence grâce à la documentation claire de l'outil et à l'environnement technique déjà bien préparé. En l'espace de quelques semaines, j'ai non seulement acquis les bases de dbt, mais j'ai aussi réussi à construire le flux SALES, composé de dix modèles dbt, tout en commençant à réfléchir aux optimisations possibles et aux perspectives d'évolution du projet.

Cette expérience a confirmé que dbt est un outil particulièrement adapté au contexte de Recommerce, une entreprise en croissance rapide, où il est crucial que de nouveaux collaborateurs puissent s'approprier les outils facilement et efficacement. dbt favorise cette montée en compétence grâce à sa logique modulaire, sa syntaxe SQL accessible, et son intégration avec des outils collaboratifs comme Git.

Par ailleurs, dbt permet également aux équipes métiers en dehors de la DIP d'accéder à des modèles ou des requêtes préparées, leur donnant plus d'autonomie pour explorer leurs propres données, sans avoir à dépendre systématiquement de l'équipe data.

En travaillant directement sur le flux SALES, j'ai pu démontrer que dbt répondait aux besoins croissants de l'équipe en termes de lisibilité, de collaboration, de maintenabilité et de scalabilité. Ces résultats concrets ont renforcé la confiance de l'équipe dans la pertinence de cet outil, justifiant ainsi la décision d'étendre progressivement son utilisation à l'ensemble des flux de données, avec une vision à long terme.

IV. Les outils utilisés

Durant mon alternance, j'ai utilisé différents outils liés à l'ingénierie des données, notamment :

- dbt : principal outil de modélisation et transformation des données,
- Snowflake : data warehouse de l'entreprise,
- DBeaver : outil de visualisation et requêtage des bases de données,
- Talend : encore utilisé pour certains flux existants non migrés,
- Jira, Confluence, Slack : outils de gestion de projet et de communication,
- VSCode : environnement de développement pour dbt.

Ces outils m'ont permis de travailler efficacement dans un environnement orienté data engineering.

A. Fonctionnement de dbt

Dans le cadre de mon alternance, dbt (data build tool) est utilisé comme principal outil de transformation des données au sein du data warehouse de l'entreprise. dbt permet d'écrire du

code SQL pour transformer les données de manière structurée, versionnée et testée, en s'appuyant sur des bonnes pratiques issues du développement logiciel. Le cœur du fonctionnement de dbt repose sur les modèles, qui sont des fichiers SQL décrivant les transformations à appliquer. Lorsqu'on exécute la commande `dbt run`, ces modèles sont compilés et exécutés pour créer des vues ou des tables dans la base de données cible. dbt permet également d'automatiser des vérifications de qualité à l'aide de tests, comme la vérification de valeurs uniques ou non nulles.

Les macros, écrites avec le langage Jinja, jouent un rôle important en permettant la création de blocs de code réutilisables, ce qui limite la redondance et facilite la standardisation des transformations. Grâce aux targets, dbt permet aussi de gérer différents environnements de travail (développement, test, production) de manière souple, en spécifiant les connexions et configurations propres à chacun. Cette organisation favorise la collaboration entre les membres de l'équipe data tout en assurant la fiabilité, la traçabilité et la documentation des transformations appliquées aux données.

B. Organisation des fichiers dans dbt

L'un des grands atouts de dbt réside dans la structuration claire de ses fichiers, qui permet une gestion rigoureuse des projets de transformation de données. Le fichier central est *dbt_project.yml*, qui définit la configuration du projet : nom, chemins d'accès aux modèles, comportements par défaut, etc. Le dossier *models* contient l'ensemble des fichiers SQL représentant les transformations. Ces modèles peuvent être organisés dans des schémas (par exemple : OPERATIONS, MARKET_DATA, MONITORING, ...), ce qui facilite la maintenance et la lisibilité du projet.

Le dossier *macros* regroupe les fonctions personnalisées écrites en Jinja, utiles pour factoriser du code ou automatiser certaines logiques complexes. Le dossier *snapshots* permet de suivre l'historique de certaines tables, en capturant les changements dans le temps pour les analyses temporelles. Le dossier *seeds* contient des fichiers CSV importables directement dans le data warehouse, souvent utilisés comme tables de référence statiques. dbt inclut aussi des dossiers pour les tests (tests de qualité des données), les docs (documentation générée automatiquement), les logs (fichiers de suivi des exécutions) et les packages (modules réutilisables partagés entre projets). Cette architecture encourage une approche modulaire, reproductible et collaborative de la gestion des données, tout en assurant une documentation intégrée et à jour.

Les informations mentionnées ci-dessus s'appuient sur la documentation officielle de dbt, disponible sur le site docs.getdbt.com, qui constitue une ressource de référence complète. Elles sont également issues de mon expérience personnelle acquise tout au long de l'année, à travers des cas concrets de modélisation, l'utilisation des macros, la gestion de plusieurs environnements (*targets*), et l'écriture de tests de validation dans un contexte professionnel.

V. Mise en place du projet

Comme mentionné précédemment, mon stage portait déjà sur DBT. Cet outil était donc déjà installé sur mon ordinateur, ainsi que GitHub, Talend et les logiciels de traitement de code tels que Visual Studio Code, DBeaver, etc.

A. Organisation

Nous avons décidé de travailler sur la base de données de CircularX. Bien que CircularX soit désormais intégré à Recommerce, les données restent séparées dans une base dédiée.

Plusieurs raisons nous ont convaincus de commencer par migrer les données de CircularX. Tout d'abord, le fait que ce soit une entreprise plus récente rend son environnement de données moins complexe que celui de Recommerce. Il y a aussi numériquement moins de données et moins de tables.

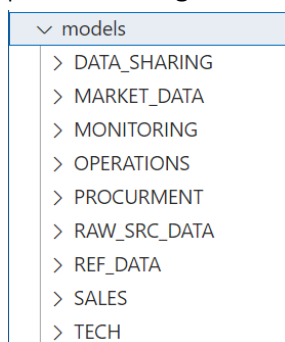
Nous avons opté pour une migration one-to-one, ce qui signifie que chaque table de l'ancienne base de données (CIRX_LIVE) aura son équivalent dans la base DBT (CIRX_LIVE_DBT), à quelques exceptions près. Nous faisons ce choix car l'environnement est déjà assez bien structuré, avec peu de dettes techniques.

De plus, cela nous permettra de faciliter les tests en comparant les données de CIRX_LIVE avec celles de CIRX_LIVE_DBT. Si les données sont identiques, cela signifie que l'extraction, la transformation et l'insertion se sont déroulées comme prévu.

C'est aussi l'occasion de supprimer les tables devenues obsolètes, par exemple celles dédiées à d'anciens clients ou des tables intermédiaires qui ne sont plus utilisées.

B. Création des modèles

Dans cette partie, nous allons nous concentrer sur la création des modèles, qui constituent la fondation de notre base de données. J'ai commencé par créer les schémas dans lesquels nous pourrions ranger les modèles.



Par exemple dans le schéma DATA_SHARING est composé de 16 modèles :

DATA_SHARING

- ✖ BBM_REPORTING.sql
- ✖ CHRISTOFLE_CUSTOMERS.sql
- ✖ CHRISTOFLE_REPORTING.sql
- ✖ DIAGNOSTIC_QUESTION_ANSWERS.sql
- ✖ REPEAT_RS_OFFER_HISTORY_MARKET_DATA.sql
- ✖ REPEAT_RS_OFFERS_MARKET_DATA.sql
- ✖ RS_BOULANGER_TRACKING.sql
- ✖ RS_BUYBACK_DETAILS.sql
- ✖ RS_CX_INVOICES_INFORMATIONS.sql
- ✖ UB_D_CATEGORY.sql
- ✖ UB_D_CUSTOMERS.sql
- ✖ UB_D_ORGANIZATION.sql
- ✖ UB_D_TRANSLATIONS.sql
- ✖ UB_F_CATALOG.sql
- ✖ UB_F_DIAGNOSTIC_DETAILS.sql
- ✖ UB_F_RACHAT.sql

Chaque modèle est constitué de 3 parties :

- La configuration (le bloc « *config* ») dans dbt permet de définir le comportement d'un modèle au moment de sa compilation et de son exécution. Elle s'écrit généralement au début d'un modèle à l'intérieur d'un bloc Jinja. Parmi les paramètres les plus couramment utilisés, on trouve *materialized*, qui détermine la manière dont le modèle est stocké (par exemple en mode table, view ou incremental), ainsi que *unique_key* et *incremental_strategy*, utiles dans les modèles incrémentaux pour identifier les lignes uniques et contrôler la logique de mise à jour. Il est également possible d'exclure certaines colonnes lors des opérations de fusion avec *merge_exclude_columns*, ou encore d'ajouter des instructions à exécuter avant ou après le modèle grâce aux options *pre_hook* et *post_hook*. Ces configurations permettent d'adapter le comportement des modèles aux besoins spécifiques du projet.
- Le ou les blocs CTE (Common Table Expressions), introduits avec le mot-clé `WITH`, permettent de structurer les transformations de données de manière lisible et modulaire. Ces blocs agissent comme des vues temporaires, définies en amont de la requête principale, et facilitent l'enchaînement logique des étapes de transformation. Ils sont souvent utilisés pour préparer des jeux de données intermédiaires, effectuer des jointures, des agrégations ou encore des nettoyages avant l'étape finale de sélection. Dans un projet dbt, chaque CTE peut représenter un sous-ensemble cohérent de données, combinant à la fois des sources brutes et d'autres modèles dbt déjà transformés. Cela permet de créer une chaîne de transformation claire, traçable et réutilisable, tout en maintenant la logique métier bien organisée. Cette pratique est particulièrement utile pour garantir la transparence des calculs, faciliter le débogage et assurer la maintenabilité des modèles sur le long terme.
- La dernière étape dans la construction d'un modèle dbt consiste généralement au formatage des données. C'est à ce moment que les types de données sont explicitement définis à l'aide du mot-clé *SELECT*, en appliquant les conversions appropriées à chaque champ. L'objectif est d'assurer que chaque colonne respecte un

format cohérent et adapté à son usage, par exemple : *VARCHAR* pour les textes, *NUMBER(15,2)* pour les montants, *TIMESTAMP_LTZ* pour les dates ou encore *BOOLEAN* pour les valeurs vraies/fausses. Ce formatage est important car il garantit que les données seront stockées correctement dans la table finale générée par dbt, qu'on appelle aussi le schéma cible, c'est-à-dire la structure de la table qui sera créée ou mise à jour dans l'entrepôt de données. Cette étape assure donc une bonne compatibilité avec les outils d'analyse ou de visualisation utilisés ensuite, et contribue à la qualité et à la fiabilité du modèle.

Pour la suite du rapport, nous prenons comme fil conducteur la création, bloc par bloc, de la table ODS_BUYBACK, présente dans le schéma OPERATIONS. Cette table a été choisie car elle illustre bien de nombreuses caractéristiques que nous avons mises en place dans nos modèles dbt.

La table ODS_BUYBACK a pour objectif principal de suivre l'ensemble du cycle de vie d'une opération de reprise (buyback) dans le système CircularX. Elle centralise les informations issues de plusieurs étapes du processus : de la création de la demande de reprise jusqu'à sa clôture, en passant par le diagnostic et la logistique.

Chaque ligne représente un rachat individuel, identifié de manière unique par un champ ID. On y retrouve des informations opérationnelles, comme le diagnostic réalisé en atelier (grade contrôlé, prix contrôlé, date de contrôle, etc.), des informations logistiques (par exemple ARRIVED_IN_STORE, WENT_THROUGH_QUARANTINE, ARRIVED_IN_WAREHOUSE), et des données organisationnelles permettant de rattacher chaque opération à une structure précise (ORGANIZATION, ROOT_ORGANIZATION, WAREHOUSE, etc.).

Certaines informations, comme le numéro d'identification, le grade déclaré ou le prix de déclaration, sont déterminées automatiquement par CircularX à partir des réponses du client à un questionnaire en ligne. Ces données sont ensuite croisées avec les référentiels envoyés par les repreneurs pour estimer une valeur de reprise. Le diagnostic final, réalisé en atelier, permet quant à lui de confirmer ou ajuster cette estimation.

La table contient également des champs techniques qui facilitent le suivi des évolutions dans le temps et la traçabilité des opérations. En résumé, ODS_BUYBACK est une table pivot, conçue pour centraliser, structurer et tracer toutes les données clés du processus de rachat, ce qui en fait un support essentiel pour les analyses de performance, la supervision opérationnelle, ainsi que le pilotage métier au sein de CircularX.

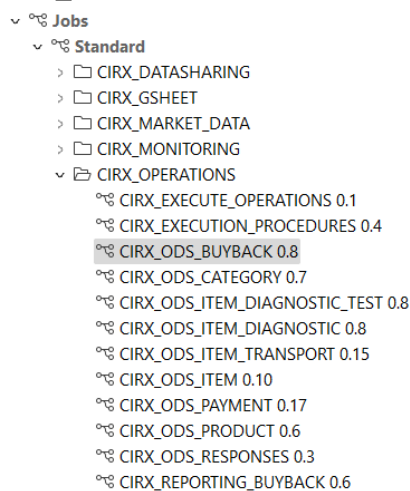
C. Construction de la table ODS_BUYBACK

Nous prenons comme exemple la table ODS_BUYBACK. Ce processus est applicable à la majorité des modèles.

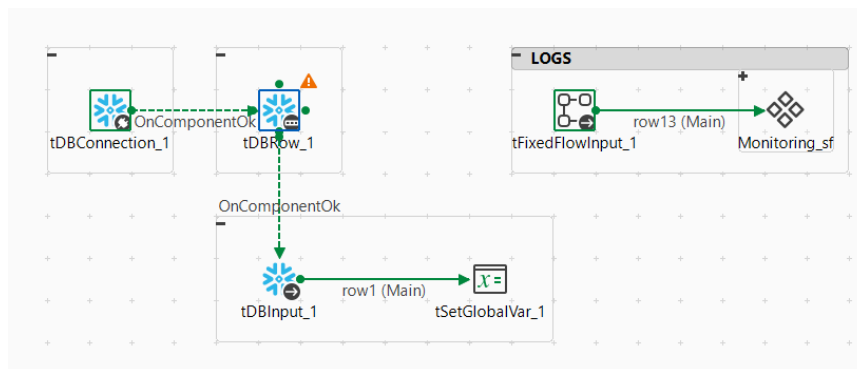
1. CTE

Nous commençons donc par identifier la requête SQL, c'est-à-dire l'endroit où se feront les transformations de données ; cette requête constituera notre bloc CTE.

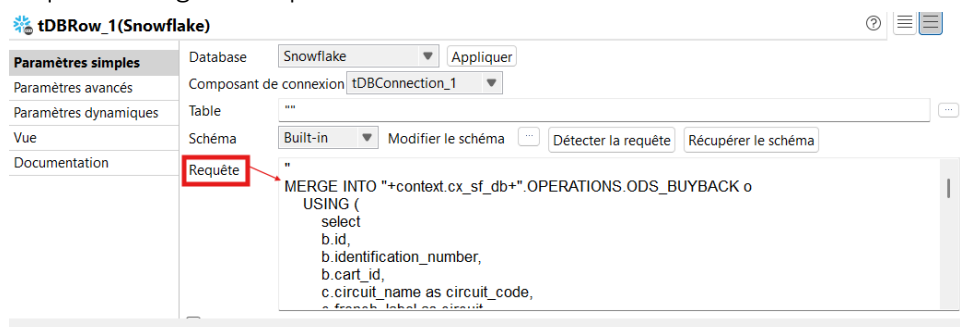
Pour cela, nous recherchons dans Talend le job responsable de la création de la table ODS_BUYBACK :



Nous accédons au job CIRX_ODS_BUYBACK, puis nous consultons la partie Designer, qui permet, via un système de glisser-déposer, d'assembler et de relier différents composants pour construire un flux de traitement de données :



Nous examinons les différents composants afin d'identifier celui dans lequel s'effectue la transformation des données. C'est à partir de ce composant que nous pourrions récupérer la requête SQL générée par Talend :



Dans notre exemple, nous trouvons la requête SQL qui crée la table ODS_BUYBACK dans le composant appelé tDBRow_1.

Dans l'annex 1 on peut retrouver la cte de cette table.

Pour cette partie, ce qui a été compliqué pour moi, c'est tout d'abord de faire le lien entre le nom de la table dans la base de données et le nom du job dans Talend, car ils ne sont souvent

pas identiques. De plus, un job Talend ne correspond pas nécessairement à une seule table dans la base ; il peut en générer plusieurs, ou au contraire, une même table peut résulter de plusieurs jobs. Il a donc fallu distinguer les différents composants d'un même job, notamment ceux contenant des requêtes SQL.

Ensuite, j'ai dû être particulièrement vigilante lors de l'intégration des requêtes dans mes modèles dbt, car les requêtes générées dans Talend utilisent une syntaxe SQL particulière. Par exemple, lorsqu'une date est utilisée, le format attendu dans Talend est le suivant :

```
""+TalendDate.formatDate("yyyy-MM-dd HH:mm:ss",(Date)
globalMap.get("last_successful_exec"))+""
```

ce qui, en SQL standard, correspondrait à : '2025-06-15 14:23:05'.

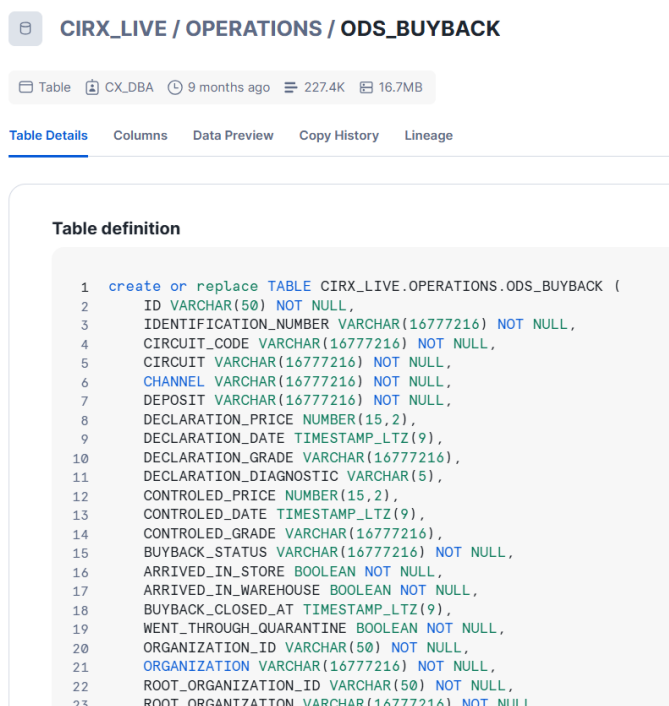
Au début, j'ai eu besoin de l'aide d'Augustin, le data manager responsable de la base de données CircularX. Aujourd'hui, je suis capable de retrouver une requête dans Talend et de l'adapter correctement pour l'intégrer dans un modèle dbt.

2. Structure de la table

Dans cette partie, nous allons retrouver tous les champs et leur type qui seront présents dans la table cible, dans notre exemple la table ODS_BUYBACK. Cependant, comme pour la partie précédente, ce processus est reproductible pour toutes les tables.

Étant donné que nous effectuons une migration one-to-one, la structure des tables dans la base de données CIRX_LIVE est identique à celle de la base que nous créons, CIRX_LIVE_DBT. Nous pouvons donc directement retrouver la structure des tables via l'interface de Snowflake.

Donc, pour la table ODS_BUYBACK, il suffit d'aller dans la base de données CIRX_LIVE, dans le schéma OPERATIONS, pour accéder directement à sa structure :



The screenshot displays the Snowflake web interface for the table **CIRX_LIVE / OPERATIONS / ODS_BUYBACK**. The table is located in the **CX_DBA** database and was last updated 9 months ago. It has a size of 227.4K and a file size of 16.7MB. The interface includes tabs for **Table Details**, **Columns**, **Data Preview**, **Copy History**, and **Lineage**. The **Table definition** tab is active, showing the following SQL code:

```
1 create or replace TABLE CIRX_LIVE.OPERATIONS.ODS_BUYBACK (
2   ID VARCHAR(50) NOT NULL,
3   IDENTIFICATION_NUMBER VARCHAR(16777216) NOT NULL,
4   CIRCUIT_CODE VARCHAR(16777216) NOT NULL,
5   CIRCUIT VARCHAR(16777216) NOT NULL,
6   CHANNEL VARCHAR(16777216) NOT NULL,
7   DEPOSIT VARCHAR(16777216) NOT NULL,
8   DECLARATION_PRICE NUMBER(15,2),
9   DECLARATION_DATE TIMESTAMP_LTZ(9),
10  DECLARATION_GRADE VARCHAR(16777216),
11  DECLARATION_DIAGNOSTIC VARCHAR(5),
12  CONTROLLED_PRICE NUMBER(15,2),
13  CONTROLLED_DATE TIMESTAMP_LTZ(9),
14  CONTROLLED_GRADE VARCHAR(16777216),
15  BUYBACK_STATUS VARCHAR(16777216) NOT NULL,
16  ARRIVED_IN_STORE BOOLEAN NOT NULL,
17  ARRIVED_IN_WAREHOUSE BOOLEAN NOT NULL,
18  BUYBACK_CLOSED_AT TIMESTAMP_LTZ(9),
19  WENT_THROUGH_QUARANTINE BOOLEAN NOT NULL,
20  ORGANIZATION_ID VARCHAR(50) NOT NULL,
21  ORGANIZATION VARCHAR(16777216) NOT NULL,
22  ROOT_ORGANIZATION_ID VARCHAR(50) NOT NULL,
23  ROOT_ORGANIZATION VARCHAR(16777216) NOT NULL,
```

Il y a toutefois un point de vigilance concernant les dates : il faut bien distinguer les dates métier, comme par exemple une date d'achat issue des tables sources, des dates techniques,

qui correspondent aux dates d'insertion ou de mise à jour dans la table cible. Ces dernières ont des noms standardisés :

- INSERTED correspond à la date de la première insertion de la donnée,
- UPDATED correspond à la date de modification, si la donnée venait à évoluer.

Si la donnée ne change jamais, alors les champs INSERTED et UPDATED auront la même valeur. De se faite dans toute les table j'ai aussi rajouter les champs techniques.

La partie de structuration de la table ODS_BUYBACK est consultable dans l'Annex 2.

3. Configuration

Dans dbt, la configuration des modèles inclut le choix du type de matérialisation, qui détermine comment les données sont stockées ou exposées. Il existe quatre principaux types de matérialisation : view, table, incremental et ephemeral.

- **View** : dbt crée une vue SQL dans la base. Il n'y a pas de logique incrémentale, la vue est recalculée à chaque requête. Ce type est adapté pour des modèles finaux destinés à la consultation simple, avec une complexité limitée et sans besoin de performances élevées. Il est utilisé lorsque le volume de données est faible ou que le modèle sert uniquement à afficher des données.
- **Table** : dbt crée une table physique et recharge complètement les données à chaque exécution, sans logique incrémentale. Ce mode convient pour des modèles finaux où la donnée doit être stockée complètement et rafraîchie intégralement. Il est utile pour des cas où les performances et l'isolement des calculs sont importants.
- **Incremental** : dbt crée une table avec une logique de chargement incrémental. Seules les nouvelles données ou celles modifiées sont chargées, ce qui est essentiel pour les gros volumes ou pour conserver un historique. Ce mode est particulièrement adapté aux tables de faits ou de dimensions dans un entrepôt de données.
- **Ephemeral** : dbt ne crée ni vue ni table pour ce modèle. Le SQL est directement intégré dans les modèles dépendants, sans stockage intermédiaire. Ce type est utilisé pour décomposer un modèle complexe en plusieurs étapes avec un scope limité, bien que son usage soit moins fréquent.

Pour la base de données de CircularX, nous avons principalement utilisé des tables de type view ou incremental. Ce choix s'explique par deux raisons principales : d'une part, l'ancienne base de données CIRX_LIVE était déjà organisée autour de ces deux types de structures, ce qui rendait naturel de conserver cette logique dans la migration. D'autre part, ces deux types de matérialisation répondent aux besoins spécifiques de performance et de maintenabilité du projet.

Les tables de type incremental sont particulièrement adaptées à la volumétrie importante de certaines tables, comme celles liées aux transactions ou aux historiques. Elles permettent de

ne recharger que les données nouvelles ou modifiées à chaque exécution, évitant ainsi de retraiter l'ensemble du jeu de données à chaque fois. Cela optimise considérablement les temps de traitement et réduit la charge sur l'entrepôt de données.

À l'inverse, les views sont utilisées pour des modèles plus légers, souvent en lecture seule, où la transformation est simple et ne nécessite pas de conservation des données. Comme elles sont recalculées à chaque requête, elles garantissent que les utilisateurs accèdent toujours à la version la plus à jour des données sources, sans avoir besoin de stocker les résultats intermédiaires.

Ainsi, en combinant judicieusement les vues pour les usages simples et les modèles incrémentaux pour les cas plus complexes ou plus lourds, nous avons pu concevoir une architecture cohérente, performante et alignée avec les pratiques déjà en place dans la base historique.

Dans la configuration, nous avons vu qu'il existait de nombreux paramètres pouvant être intégrés. Je ne les détaillerai pas ici, car cette partie se concentre sur la base d'un modèle. À partir du moment où l'on précise le type de table dans la configuration, et que les autres sections du modèle sont correctement complétées, celui-ci devient opérationnel et renvoie des résultats. (voir annexe 3 pour la config de la table ODS_BUYBACK)

VI. L'évolution des model au cours de l'alternance

De nombreuses améliorations ont été apportées tout au long de l'année pour répondre à différents besoins, et celles-ci feront l'objet des parties suivantes.

A. Incrementation

Nous avons vu que dans la base de donner de cricularx les deux types de table utiliser été dont les tables incrémentales. Dans dbt, les modèles incrémentaux permettent d'optimiser les performances en évitant de recharger l'intégralité d'une table à chaque exécution. Au lieu de cela, seuls les nouveaux enregistrements ou les enregistrements modifiés sont traités et intégrés. Cette approche est particulièrement adaptée aux volumes de données importants ou aux processus de mise à jour fréquente.

Plusieurs stratégies d'incrémentation peuvent être configurées selon le comportement souhaité et les capacités du data warehouse utilisé :

- **append** : ajoute simplement les nouvelles lignes sans modifier les anciennes. C'est la stratégie la plus simple mais elle nécessite que les données soient strictement ajoutées, sans modification.
- **delete+insert** : supprime puis réinsère les lignes correspondant à une clé unique (unique_key). Cela permet de gérer les modifications mais peut être coûteux si les suppressions sont massives.
- **merge** : met à jour les lignes existantes sur la base de la unique_key et insère les nouvelles lignes. C'est la stratégie la plus flexible pour gérer les modifications incrémentales. Il est également possible de spécifier les colonnes à mettre à jour ou à exclure avec merge_update_columns ou merge_exclude_columns.

Le choix de la stratégie dépend donc de plusieurs facteurs : la volumétrie, la structure des données, la fiabilité des clés uniques et les fonctionnalités supportées par la plateforme (par exemple, Snowflake prend en charge toutes les stratégies tandis que BigQuery n'en supporte que certaines).

1. Mise en place du merge

Ce sujet a nécessité une réflexion collective au sein de l'équipe. Initialement, nous nous étions orientés vers une stratégie de type delete+insert. Cependant, lors des premiers tests sur la base de données, plusieurs problèmes sont apparus : certaines tables ne disposaient pas de clé primaire suffisamment fiable, et nous avons constaté des incohérences dans les données sources. Cela s'explique en partie par le changement d'architecture : auparavant, Talend assurait l'extraction directe des données depuis les applications, tandis qu'avec dbt, cette phase d'extraction n'est plus prise en charge. Nous avons donc dû tester d'autres outils pour la partie "extraction", ce qui a pu introduire des erreurs dans les données durant la phase de test. Au vu de ces constats, nous avons finalement décidé d'opter pour la stratégie incrémentale merge, plus adaptée à la gestion de données susceptibles de contenir des mises à jour.

Dans le cadre de la modélisation de la table ODS_BUYBACK, nous avons opté pour une stratégie incrémentale basée sur la méthode merge, car elle s'avère particulièrement adaptée à notre cas d'usage. En effet, les données issues des diagnostics (web, magasin, entrepôt) peuvent évoluer dans le temps, tout comme le statut du rachat ou les informations d'organisation. Une simple stratégie de type insert n'aurait pas permis de refléter ces mises à jour dans la base cible.

La stratégie merge permet de détecter les changements au niveau des lignes existantes (grâce à la clé unique ID) et de mettre à jour les champs modifiés tout en insérant les nouvelles lignes. Cela garantit une meilleure fraîcheur des données et évite les duplications, tout en assurant une traçabilité temporelle via les champs INSERTED et UPDATED. Cette approche est d'autant plus pertinente dans un contexte où les données sources peuvent contenir des corrections ou être soumises à des rechargements partiels, notamment en phase de transition d'un outil ETL (Talend) vers dbt. Par ailleurs, nous avons choisi d'ajouter le champ INSERTED au paramètre merge_exclude_columns, afin de préserver la date d'insertion initiale d'une donnée. Cette précaution permet d'éviter qu'elle soit écrasée lors d'une mise à jour via le merge, et ainsi de toujours pouvoir identifier le moment exact où une donnée est apparue pour la première fois dans la table.

2. Quelle que exception

Il y a cependant 2 tables dans la base de données de circularx pour lesquelles la stratégie append est plus adaptée : BI_REEPEAT_OFFERS_DAILY_PRICE et PRODUCT_PRICES. Ces tables sont conçues pour enregistrer des instantanés journaliers ou des états successifs des prix des offres. Dans ces cas, chaque nouvelle ligne représente une valeur historique distincte (par exemple, un prix observé à une date donnée pour une offre donnée, ou un prix par grade pour un produit à un instant t). Il ne s'agit pas de mettre à jour des lignes existantes, mais bien d'accumuler des observations dans le temps. C'est pourquoi la stratégie append, qui ajoute simplement les

nouvelles données sans chercher à déduplicer ou à mettre à jour les anciennes, est ici bien plus appropriée. Elle permet de préserver l'historique complet sans risque d'écraser des données précédemment insérées, contrairement à la stratégie merge qui est mieux adaptée aux cas où l'on cherche à maintenir une version actualisée d'un même enregistrement.

3. Macros

Dans les modèles de merge de notre projet dbt, nous avons réutilisé une macro que j'ai développée lors de mon stage : `filter_by_max_date`. Cette macro est conçue pour optimiser l'incrémentation des données en se basant sur des champs date et pas que rapport à l'ID.

La macro commence par identifier la table concernée, puis récupère la date maximale présente dans chacun des deux champs si les deux sont fournis. Elle compare ensuite ces deux dates et conserve la plus récente comme référence. Un paramètre de sécurité, `days_ago`, défini par défaut à -1 dans le fichier `dbt_project.yml`, permet d'ajuster cette date en la reculant d'un jour. Cette logique vise à garantir qu'aucune donnée limite ne soit accidentellement ignorée, notamment en cas de décalage ou de latence dans l'ingestion des données. La macro génère alors dynamiquement une clause SQL qui filtre les enregistrements en ne conservant que ceux dont les dates sont supérieures ou égales à la date ajustée.

Voici un exemple d'utilisation dans un modèle : la macro est appelée à la fin de la requête principale pour filtrer efficacement les lignes à traiter. Grâce à cette implémentation, nous mutualisons une logique d'incrémentation robuste, flexible et sécurisée sur l'ensemble de nos modèles dbt. En cas de besoin, la variable `days_ago` peut être modifiée ponctuellement pour réintégrer des données plus anciennes, par exemple dans le cadre d'un refresh exceptionnel ou d'un correctif. Le code de la macros et sont appelle dans la table ODS_BUYBACK sont dans l'annex 4.

B. Gestion des sources

Pour construire les modèles dans dbt, nous nous appuyons sur différentes sources de données. Ces données peuvent provenir soit de tables brutes issues directement des applications (raw data), soit d'autres modèles dbt déjà construits. Afin de permettre à dbt de comprendre la hiérarchie entre les tables et de gérer correctement l'ordre d'exécution, il est nécessaire d'adapter la syntaxe des appels de tables dans les requêtes, notamment dans les clauses FROM.

Lorsqu'un modèle utilise les données d'un autre modèle dbt, on utilise la fonction `ref()`. Par exemple, si le modèle `ODS_BUYBACK` dépend du modèle `ODS_ITEM_DIAGNOSTIC`, au lieu d'écrire directement `CIRX_LIVE_DBT.OPERATION.ODS_ITEM_DIAGNOSTIC`, on utilisera la syntaxe dbt suivante :

```
{{ ref('ODS_ITEM_DIAGNOSTIC') }}
```

Cette écriture indique à dbt qu'il doit exécuter d'abord le modèle `ODS_ITEM_DIAGNOSTIC` avant `ODS_BUYBACK`, respectant ainsi les dépendances entre les modèles.

En revanche, lorsque le modèle exploite des données brutes (non transformées), il ne s'agit pas de modèles dbt, mais de sources. Dans ce cas, on utilise la fonction `source()` après avoir déclaré

les tables concernées dans un fichier `sources_raw_data.yml` (voir annexe 5). Par exemple, dans le modèle `ODS_BUYBACK`, un appel à une table source ressemblera à :

```
{{ source('raw_src', 'circularx_stock_buyback') }}
```

L'utilisation systématique de `ref()` et `source()` permet à dbt de déterminer automatiquement l'ordre d'exécution des modèles et des sources. Grâce à cela, nous pouvons visualiser clairement le flux de transformation des données dans l'interface dbt, notamment via la fonctionnalité lineage (voir annexe 6), qui montre les dépendances entre toutes les tables du projet.

C. Model python

Concernant les modèles Python, bien qu'ils ne soient pas encore intégrés dans notre pipeline principal, j'ai pu expérimenter leur création dans le cadre de mes recherches. J'ai notamment réussi à développer des modèles très simples en Python (voir Annex 7), qui s'exécutent correctement dans dbt. Toutefois, ces modèles Python génèrent actuellement des tables via des scripts exécutés directement dans Snowflake, et non dans l'outil Talend. Par conséquent, leur intégration est considérée comme moins prioritaire à court terme, comparée aux tables critiques générées par Talend que nous devons migrer en priorité. Néanmoins, cette piste reste prometteuse, car elle ouvre la voie à une centralisation des transformations de données au sein de dbt, en utilisant exclusivement Python. Cela pourrait permettre, à terme, une meilleure cohérence technique et une maintenance facilitée du projet.

D. Intégration des tests de qualité des données

dbt fournit nativement plusieurs tests génériques, tels que `unique`, `not_null`, `accepted_values` ou encore `relationships`, permettant de valider l'intégrité des modèles en s'assurant par exemple qu'une colonne ne contient pas de doublons ou de valeurs nulles. Il est également possible de créer des tests personnalisés en SQL (appelés `singular tests`), ou même d'utiliser des tests en Python, notamment avec `Snowpark`, pour des cas plus complexes ou spécifiques. L'objectif est d'intégrer ces tests tout au long du pipeline de traitement afin de garantir la qualité des données à chaque étape de l'exécution.

Il est également envisageable d'associer des niveaux de gravité à chaque test. Par exemple, un test critique peut provoquer l'arrêt complet du pipeline en cas d'échec, tandis qu'un test moins important pourra simplement générer un avertissement. Cela permet de mieux prioriser les actions en fonction de l'impact réel sur la qualité globale des données.

Pour le moment, les tests ne sont pas encore intégrés dans le projet dbt, car ils seront incorporés dans l'outil de mise en production `Prefect`, qui est en cours de mise en place et d'adaptation à la base de données. Cela permettra de déclencher les tests à des moments stratégiques dans le pipeline et dans un ordre logique, garantissant ainsi une meilleure maîtrise de la qualité à mesure que les transformations avancent.

E. Procedure

Les procédures Python dans Snowflake permettent d'exécuter du code Python directement dans l'environnement Snowflake, au plus près des données, sans avoir à les extraire dans un outil externe. Elles sont particulièrement utiles pour automatiser des traitements complexes, effectuer des transformations avancées ou encore appliquer des règles métier spécifiques, difficiles à exprimer en SQL pur. Cela devient encore plus pertinent dans le cadre de projets dbt, où il est possible d'appeler ces procédures directement dans la configuration des modèles à l'aide des hooks. Par exemple, dans la configuration du modèle ODS_BUYBACK, on peut inclure plusieurs procédures Python dans le `pre_hook`, comme `STATUS_HISTO_DESERIALIZATION`, `UNSERIALIZE_CARRIER_STATUS` ou encore `HISTO_STATUS_TO_JSON_FORMAT`. Ces procédures permettent de préparer ou enrichir les données en amont de l'exécution du modèle. On peut également y inclure la création dynamique d'une table si elle n'existe pas encore, tout cela étant géré automatiquement avant le chargement des données. Pour faciliter l'appel de ces procédures dans différents modèles, j'ai mis en place une macro standardisée `call_proc` (voir Annexe 7), qui permet de centraliser et simplifier l'exécution des procédures en dbt, tout en assurant une meilleure lisibilité et maintenance du code. Cette intégration renforce la puissance de dbt et permet d'assurer une meilleure orchestration des étapes critiques du pipeline de données.

VII. Suite du projet avec la base de données Recommerce

Aujourd'hui, la base de données de Circulax est quasiment prête à être mise en production. Cela nous a permis de commencer à préparer le terrain pour la future base de données de Recommerce. Plutôt que d'envisager une migration one-to-one, nous avons saisi cette opportunité pour repenser et optimiser l'architecture existante. L'objectif est de corriger la dette technique accumulée, de scinder certaines tables devenues trop complexes, de supprimer celles qui sont désormais obsolètes et d'en créer de nouvelles pour répondre aux besoins émergents.

1. guidelines

Dans cette dynamique, nous avons commencé à définir des bonnes pratiques et à instaurer une standardisation globale de l'écosystème data. Plusieurs axes ont été définis : des guidelines claires sur la gestion des nombres (ex. : `Number(20,4)` pour les données brutes, `Number(10,2)` pour les montants, `Number(10,4)` pour les taux de change), la gestion des champs nuls dépendant des règles métier, l'usage des fonctions de division sécurisée comme `div0`, ou encore la normalisation des dates avec des champs `INSERTED` et `MODIFIED` forcés en `current_timestamp` et des noms de champs source préfixés par `src_`. Tous les champs temporels doivent être en LTZ (Local Time Zone). Au niveau du nommage, les modèles BI doivent refléter le nom du processus métier, les modèles ODS porter le suffixe `_ODS`, et les tables temporaires être identifiées par le suffixe `_temp`. Ces dernières doivent être facilement identifiables puisqu'elles n'ont pas de finalité métier directe, mais sont essentielles dans la chaîne de traitement. Enfin, certaines règles sont rendues obligatoires dans tous les modèles : bloc config, stratégie incremental avec merge, usage systématique des CTEs, références explicites aux autres modèles dbt, et présence d'un identifiant unique pour chaque table, tant

en ODS qu'en BI. Cette nouvelle base sera donc plus robuste, plus lisible, et mieux adaptée aux évolutions futures.

Les source.

2. Les source

Pour initier la construction de la nouvelle base de données Recommerce, il a d'abord été nécessaire de recenser l'ensemble des sources de données exploitées dans l'écosystème existant. Les données utilisées dans les modèles Recommerce proviennent d'une grande diversité d'applications et de serveurs, parfois hétérogènes, ce qui complexifie leur gouvernance. Ce travail d'inventaire a été réalisé à partir de l'outil Talend, en parcourant tous les jobs d'intégration de données et en identifiant les connexions, bases sources, tables, et leurs usages métiers. J'ai compilé ces informations dans un fichier Excel d'environ 1400 lignes, structuré avec des colonnes telles que le Projet, le Domaine fonctionnel, le Sous-domaine, le Nom du job Talend, le Type de source (Gsheet, MySQL, etc.), le Nom de la connexion, la base de données source, la table source, mais aussi des informations de gouvernance comme le responsable métier, le statut de la table (à garder ou non), son niveau de priorisation, son rôle dans le legacy Raw ou BI, et les impacts métiers éventuels. Certaines lignes comportent également des commentaires fonctionnels, ou des liens vers des tickets Jira lorsqu'un sujet d'intégration est en cours ou prévu. Ce fichier constitue aujourd'hui une base de référence essentielle pour organiser la migration, identifier les chantiers de refonte, repérer les doublons ou redondances, et structurer progressivement un environnement plus propre, mieux documenté et plus adapté aux standards futurs.

3. Repartition des rôles

Dans le cadre de l'organisation de la migration vers la nouvelle base Recommerce, nous avons choisi de répartir les responsabilités au sein de l'équipe DIP en fonction des compétences et connaissances métier de chacun. Concrètement, chaque membre de l'équipe se verra attribuer un ou plusieurs schémas correspondant à un domaine fonctionnel spécifique (ex. SALES, PRICING, FINANCE), avec pour mission de structurer les tables, clarifier les besoins métiers et assurer leur cohérence. Certains collaborateurs, davantage orientés métier, auront pour rôle principal d'exprimer les besoins, de nommer correctement les tables et de valider la logique fonctionnelle. D'autres, comme moi, plus à l'aise techniquement, seront chargés de modéliser les tables dans DBT, d'implémenter la logique de transformation, de rédiger les tests, et d'automatiser les workflows. Enfin, certains membres pourront intervenir sur les deux aspects, à la fois technique et fonctionnel, ce qui favorisera les échanges transverses et la fluidité dans la gestion des projets.

Nous avons pour objectif de finaliser cette migration avant le 15 novembre, date à laquelle la licence Talend actuelle expirera et ne sera pas renouvelée. Cette échéance impose un rythme soutenu, avec une coordination rigoureuse entre les différents rôles pour garantir la continuité des flux de données et une mise en production sans rupture.

VIII. Conclusion

En conclusion, cette phase de migration vers la nouvelle base Recommerce représente un tournant stratégique pour l'unification, la fiabilisation et l'industrialisation de nos données. Grâce à une répartition claire des rôles au sein de l'équipe DIP, nous avons pu avancer efficacement tout en valorisant les expertises de chacun, qu'elles soient métiers, techniques ou hybrides. À ce jour, 95 modèles ont été créés et sont pleinement fonctionnels, générant des pipelines stables et exécutables automatiquement. Des documentations ont également été produites pour accompagner la compréhension et l'appropriation des nouvelles notions introduites (bonnes pratiques, standardisation, architecture cible, etc.).

Au-delà des aspects techniques, j'ai accordé une grande importance à la professionnalisation de ma démarche et à la qualité de la communication au sein de l'équipe. Cela m'a permis de mieux collaborer, d'anticiper les besoins des autres membres et de contribuer activement à une dynamique collective saine et productive.

Enfin, nous restons pleinement mobilisés autour de l'échéance du 15 novembre, date de fin de la licence Talend, avec l'objectif clair de finaliser l'ensemble des travaux d'ici là. Cette contrainte temporelle nous pousse à maintenir un cap rigoureux tout en garantissant la qualité et la pérennité de notre nouvelle infrastructure data.

Si j'avais eu la possibilité de rester au sein de l'entreprise après le mois d'août, j'aurais souhaité approfondir davantage la phase de recette métier avec les utilisateurs finaux. Une implication plus précoce de leur part dans la validation des modèles aurait permis de consolider encore davantage la fiabilité des transformations et de gagner du temps en phase de mise en production. Cette perspective montre à quel point le dialogue entre technique et métier reste un facteur clé de réussite pour les projets data à fort enjeu.

IX. Glossair

- DIP : Il s'agit du service où j'ai travaillé durant mon stage. L'acronyme signifie Data, Intelligence & Pricing. Il est en charge de la plateforme analytique de Recommerce, du traitement des données jusqu'à leur valorisation.
- dbt (Data Build Tool) : Un outil open-source de transformation des données qui permet de centraliser, modéliser et versionner le code SQL de manière collaborative. Il facilite la construction de modèles de données reproductibles et maintenables.
- Data Engineering : La discipline consistant à concevoir, construire et maintenir les infrastructures et les pipelines nécessaires à la collecte, la transformation et la mise à disposition des données.
- ODS (Operational Data Store) : Un niveau de stockage intermédiaire entre les systèmes sources et les systèmes décisionnels. Il permet de stocker des données opérationnelles nettoyées et historisées, prêtes pour des traitements analytiques.
- BI (Business Intelligence) : L'ensemble des technologies, méthodes et outils permettant d'analyser les données d'une entreprise afin de soutenir les prises de décisions stratégiques et opérationnelles.
- Job : Une séquence automatisée de tâches de traitement de données, souvent orchestrée dans des outils comme Talend ou Airflow, visant à répondre à un besoin fonctionnel ou technique.
- Pipeline : Un enchaînement structuré de traitements automatiques permettant de déplacer, transformer ou enrichir des données depuis une source jusqu'à leur destination finale. Un pipeline peut comporter plusieurs étapes (extraction, transformation, chargement).
- Flux : Un terme plus général désignant la circulation des données à travers différents systèmes ou étapes de traitement. Il peut inclure des pipelines mais aussi des processus manuels ou semi-automatisés.
- Modèle (Model) : Dans le contexte de dbt, un modèle est un fichier SQL représentant une transformation de données. Il sert à créer des vues ou des tables matérialisées dans l'entrepôt de données.
- SQL (Structured Query Language) : Un langage informatique utilisé pour interagir avec les bases de données relationnelles. Il permet d'interroger, insérer, mettre à jour ou supprimer des données, mais aussi de créer des structures comme des tables, vues ou procédures.

X. Webographie

<https://www.recommerce.com/fr/qui-est-recommerce>

<https://www.rachatdemobile.com/recycleurs-mobiles/monextel>

<https://www.lejdd.fr/economie/le-reconditionne-un-marche-en-pleine-expansion-149690>

<https://economie-circulaire.ademe.fr/economie-circulaire>

<https://www.ecoco2.com/blog/le-recommerce-une-tendance-qui-se-confirme/>

XI. Annexes

Annexe 1 :

```
WITH ODS_BUYBACK AS(
select
    b.id,
    b.identification_number,
    b.cart_id,
    c.circuit_name as circuit_code,
    c.french_label as circuit,
    c.channel,
    c.deposit,
    -- Déclaration
    coalesce(ids.diagnostic_price, idw.diagnostic_price) as declaration_price,
    coalesce(ids.created_at, idw.created_at) as declaration_date,
    coalesce(ids.product_state, idw.product_state) as declaration_grade,
    case when ids.diagnostic_price is null then 'Web' else 'Store' end as declaration_diagnostic,
    -- Controle
    ida.diagnostic_price as controled_price,
    ida.created_at as controled_date,
    ida.product_state as controled_grade,
    -- State
    b.status as buyback_status,
    ids.diagnostic_price is not null as arrived_in_store,
    ida.diagnostic_price is not null as arrived_in_warehouse,
    bc.logged_at as buyback_closed_at,
    iq.id is not null as went_through_quarantine,
    -- Organization
    b.organization_id,
    o.common_name as organization,
    b.external_id,
    b.root_organization_id,
    ro.common_name as root_organization,
    i.warehouse_id,
    i.warehouse_name,
    i.buyer_account_id as LCM_ORGANIZATION_ID,
    lo.common_name as lcm_organization,
    -- update
    b.created_at as buyback_created_at, b.updated_at as buyback_updated_at,
    g.identification_number is not null as is_golden_order
from {{ source('raw_src', 'circularx_stock_buyback') }} b
inner join {{ source('raw_src', 'circularx_stock_item') }} i on i.buyback_id = b.id
inner join {{ source('raw_src', 'gsheet_circuit_dim') }} c on c.id = b.circuit_id
inner join {{ source('raw_src', 'circularx_stock_payment') }} p on p.id = b.payment_id
inner join {{ source('raw_src', 'circularx_stock_organization') }} o on o.id = b.organization_id
inner join {{ source('raw_src', 'circularx_stock_organization') }} ro on ro.id = b.root_organization_id
left join {{ source('raw_src', 'circularx_stock_organization') }} lo on lo.id = i.buyer_account_id
left join (select * from {{ ref('ODS_ITEM_DIAGNOSTIC') }} where diagnostic_type like 'seller') idw on idw.item_id = i.id
left join (select * from {{ ref('ODS_ITEM_DIAGNOSTIC') }} where diagnostic_type like 'store') ids on ids.item_id = i.id
left join (select * from {{ ref('ODS_ITEM_DIAGNOSTIC') }} where diagnostic_type like 'warehouse') ida on ida.item_id = i.id
left join (select item_id, max(logged_at) as logged_at from operations.status_history where status in ('bb_cancelled', 'bb_completed', 'bb_not_completed', 'bb_completed_with_negociation',
left join (select id from {{ source('raw_src', 'circularx_stock_item') }} where status_date like '%item_in_quarantine%') iq on iq.id = i.id
left join {{ source('raw_src', 'gsheet_golden_order') }} g on g.identification_number = b.identification_number
```

Annexe 2 :

```

)
SELECT
  ID::VARCHAR AS ID,
  IDENTIFICATION_NUMBER::VARCHAR AS IDENTIFICATION_NUMBER,
  CIRCUIT_CODE::VARCHAR AS CIRCUIT_CODE,
  CIRCUIT::VARCHAR AS CIRCUIT,
  CHANNEL::VARCHAR AS CHANNEL,
  DEPOSIT::VARCHAR AS DEPOSIT,
  DECLARATION_PRICE::NUMBER(15,2) AS DECLARATION_PRICE,
  DECLARATION_DATE::TIMESTAMP_LTZ AS DECLARATION_DATE,
  DECLARATION_GRADE::VARCHAR AS DECLARATION_GRADE,
  DECLARATION_DIAGNOSTIC::VARCHAR AS DECLARATION_DIAGNOSTIC,
  CONTROLLED_PRICE::NUMBER(15,2) AS CONTROLLED_PRICE,
  CONTROLLED_DATE::TIMESTAMP_LTZ AS CONTROLLED_DATE,
  CONTROLLED_GRADE::VARCHAR AS CONTROLLED_GRADE,
  BUYBACK_STATUS::VARCHAR AS BUYBACK_STATUS,
  ARRIVED_IN_STORE::BOOLEAN AS ARRIVED_IN_STORE,
  ARRIVED_IN_WAREHOUSE::BOOLEAN AS ARRIVED_IN_WAREHOUSE,
  BUYBACK_CLOSED_AT::TIMESTAMP_LTZ AS BUYBACK_CLOSED_AT,
  WENT_THROUGH_QUARANTINE::BOOLEAN AS WENT_THROUGH_QUARANTINE,
  ORGANIZATION_ID::VARCHAR AS ORGANIZATION_ID,
  ORGANIZATION::VARCHAR AS ORGANIZATION,
  ROOT_ORGANIZATION_ID::VARCHAR AS ROOT_ORGANIZATION_ID,
  ROOT_ORGANIZATION::VARCHAR AS ROOT_ORGANIZATION,
  WAREHOUSE_ID::VARCHAR AS WAREHOUSE_ID,
  WAREHOUSE_NAME::VARCHAR AS WAREHOUSE_NAME,
  LCM_ORGANIZATION_ID::VARCHAR AS LCM_ORGANIZATION_ID,
  LCM_ORGANIZATION::VARCHAR AS LCM_ORGANIZATION,
  BUYBACK_CREATED_AT::TIMESTAMP_LTZ AS BUYBACK_CREATED_AT,
  EXTERNAL_ID::VARCHAR AS EXTERNAL_ID,
  IS_GOLDEN_ORDER::BOOLEAN AS IS_GOLDEN_ORDER,
  BUYBACK_UPDATED_AT::TIMESTAMP_LTZ AS BUYBACK_UPDATED_AT,
  CART_ID::VARCHAR AS CART_ID,
  CURRENT_TIMESTAMP()::TIMESTAMP_LTZ AS INSERTED,
  CURRENT_TIMESTAMP()::TIMESTAMP_LTZ AS UPDATED
FROM ODS_BUYBACK

```

Annexe 3 :

```

{{
  config(
    materialized = 'incremental',
    unique_key='ID',
    merge_exclude_columns = ['INSERTED'],
    incremental_strategy = 'merge',
    pre_hook =[
      "{{ call_proc(proc_name='STATUS_HISTO_DESERIALIZATION', schema='OPERATIONS', db='CIRX_LIVE_DBT') }}",
      "{{ call_proc(proc_name='UNSERIALIZE_CARRIER_STATUS', schema='OPERATIONS', db='CIRX_LIVE_DBT') }}",
      "{{ call_proc(proc_name='HISTO_STATUS_TO_JSON_FORMAT', schema='OPERATIONS', db='CIRX_LIVE_DBT') }}",
      ""
    ]
  )
  create TABLE IF NOT EXISTS {{this.schema}}.ODS_BUYBACK (
    ID VARCHAR(50) NOT NULL,
    IDENTIFICATION_NUMBER VARCHAR() NOT NULL,
    CIRCUIT_CODE VARCHAR() NOT NULL,
    CIRCUIT VARCHAR() NOT NULL,
    CHANNEL VARCHAR() NOT NULL,
    DEPOSIT VARCHAR() NOT NULL,
    DECLARATION_PRICE NUMBER(15,2),
    DECLARATION_DATE TIMESTAMP_LTZ(9),
    DECLARATION_GRADE VARCHAR(),
    DECLARATION_DIAGNOSTIC VARCHAR(5),
    CONTROLLED_PRICE NUMBER(15,2),
    CONTROLLED_DATE TIMESTAMP_LTZ(9),
    CONTROLLED_GRADE VARCHAR(),
    BUYBACK_STATUS VARCHAR() NOT NULL,
    ARRIVED_IN_STORE BOOLEAN NOT NULL,
    ARRIVED_IN_WAREHOUSE BOOLEAN NOT NULL,
    BUYBACK_CLOSED_AT TIMESTAMP_LTZ(9),
    WENT_THROUGH_QUARANTINE BOOLEAN NOT NULL,
    ORGANIZATION_ID VARCHAR(50) NOT NULL,
    ORGANIZATION VARCHAR() NOT NULL,
    ROOT_ORGANIZATION_ID VARCHAR(50) NOT NULL,
    ROOT_ORGANIZATION VARCHAR() NOT NULL,
    WAREHOUSE_ID VARCHAR(50),
    WAREHOUSE_NAME VARCHAR(50),
    LCM_ORGANIZATION_ID VARCHAR(50),
    LCM_ORGANIZATION VARCHAR(),
    BUYBACK_CREATED_AT TIMESTAMP_LTZ(9),
    EXTERNAL_ID VARCHAR(),
    IS_GOLDEN_ORDER BOOLEAN DEFAULT FALSE,
    INSERTED TIMESTAMP_LTZ(9) DEFAULT CURRENT_TIMESTAMP(),
    UPDATED TIMESTAMP_LTZ(9),
    BUYBACK_UPDATED_AT TIMESTAMP_LTZ(9),
    CART_ID VARCHAR(50),
    primary key (ID)
  )
  ""
}
}
}}

```

Annexe 4 :

```

{% macro filter_by_max_date(champ1, champ2=None) %}

    -- Déterminer la table actuelle
    {% set table_name = this.schema ~ '.' ~ this.table %}

    -- Ajout des tags du model
    {% do set_query_tag() %}

    -- Récupérer la dernière valeur du champ1 (date principale)
    {% set query1 = "SELECT MAX(" ~ champ1 ~ ") FROM " ~ table_name %}
    {% set max_value1_result = run_query(query1) %}
    {% set max_value1 = max_value1_result.rows[0][0] if max_value1_result and max_value1_result.rows else None %}

    -- Récupérer la dernière valeur du champ2 (si fourni)
    {% if champ2 %}
        {% set query2 = "SELECT MAX(" ~ champ2 ~ ") FROM " ~ table_name %}
        {% set max_value2_result = run_query(query2) %}
        {% set max_value2 = max_value2_result.rows[0][0] if max_value2_result and max_value2_result.rows else None %}
    {% else %}
        {% set max_value2 = None %}
    {% endif %}

    -- Comparer les deux dates pour garder la plus grande
    {% if max_value1 and max_value2 %}
        {% set max_value = max_value1 if max_value1 >= max_value2 else max_value2 %}
    {% else %}
        {% set max_value = max_value1 if max_value1 else max_value2 %}
    {% endif %}

    -- Valeur par défaut si aucune date trouvée
    {% set date_incr = max_value if max_value else "1900-01-01" %}

    -- Gestion du décalage en jours
    {% set days_offset = var('days_ago') %}
    {% set filter_date = "DATEADD(day, " ~ days_offset ~ ", '" ~ date_incr ~ "'" %}

    -- Génération de la condition SQL
    (
        {{ champ1 }} >= {{ filter_date }}
        {% if champ2 %}
            OR {{ champ2 }} >= {{ filter_date }}
        {% endif %}
    )

{% endmacro %}

```

Annexe 5 :

```

models > ! sources_raw_data.yaml
1  version: 2
2  sources:
3    - name: raw_src
4      schema: RAW_SRC_DATA
5      tables:
6        Generate model
7        - name: gsheets_traductions
8          description: ""
9        Generate model
10       - name: circularx_stock_stock
11         description: ""
12       Generate model
13       - name: reepeat_mongo_offers_history
14         description: ""
15       Generate model
16       - name: reepeat_mongo_transactions
17         description: ""
18       Generate model
19       - name: reepeat_mongo_offers
20         description: ""
21       Generate model
22       - name: reepeat_mongo_products
23         description: ""
24       Generate model
25       - name: reepeat_category
26         description: ""
27       Generate model
28       - name: gsheets_edel_credit_history
29         description: ""
30       Generate model
31       - name: circularx_stock_buyer
32         description: ""
33       Generate model
34       - name: GSHEET_MANGOPAY_CREDIT_HISTORY
35         description: ""
36       Generate model
37       - name: dates_d
38         description: ""
39       Generate model
40       - name: circularx_stock_category_dispatch_circuit_rule
41         description: ""
42       Generate model
43       - name: circularx_stock_dispatch_circuit_rule
44         description: ""
45       Generate model
46       - name: gsheets_golden_order
47         description: ""
48       Generate model
49       - name: circularx_stock_item_diagnostic
50         description: ""
51       Generate model
52       - name: circularx_stock_circuit
53         description: ""
54       Generate model
55       - name: circularx_stock_transport
56         description: ""

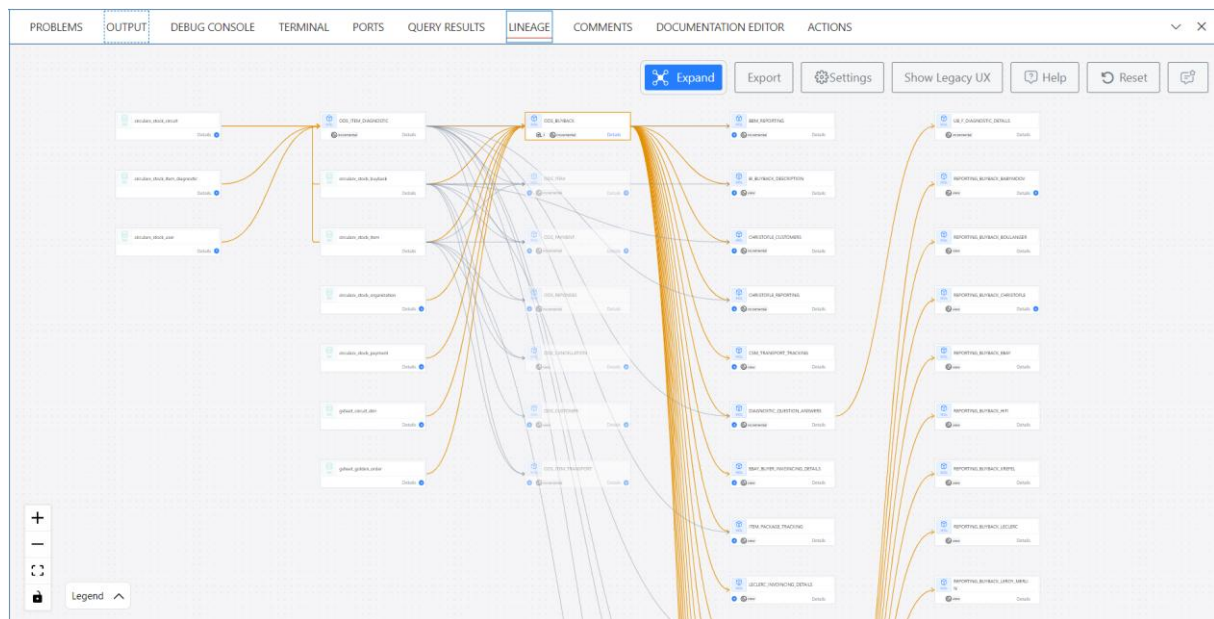
```

```

44 | | | - name: circularx_stock_payment
45 | | | | description: ""
    | | | Generate model
46 | | | - name: circularx_stock_payment_mode
47 | | | | description: ""
    | | | Generate model
48 | | | - name: circularx_stock_buyback
49 | | | | description: ""
    | | | Generate model
50 | | | - name: gsheets_circuit_dim
51 | | | | description: ""
    | | | Generate model
52 | | | - name: circularx_stock_product
53 | | | | description: ""
    | | | Generate model
54 | | | - name: circularx_stock_category
55 | | | | description: ""
    | | | Generate model
56 | | | - name: circularx_stock_item
57 | | | | description: ""
    | | | Generate model
58 | | | - name: circularx_stock_buyback_selected_response
59 | | | | description: ""
    | | | Generate model
60 | | | - name: circularx_stock_question
61 | | | | description: ""
    | | | Generate model
62 | | | - name: manual_question_categorisation
63 | | | | description: ""
    | | | Generate model
64 | | | - name: circularx_stock_user
65 | | | | description: ""
    | | | Generate model
66 | | | - name: circularx_stock_organization
67 | | | | description: ""
    | | | Generate model
68 | | | - name: py_zoho_invoices
69 | | | | description: ""
    | | | Generate model
70 | | | - name: circularx_stock_product_category
71 | | | | description: ""
    | | | Generate model
72 | | | - name: s3_gamecash_ebay_catalog
73 | | | | description: ""
    | | | Generate model
74 | | | - name: s3_kazoo_ebay_catalog
75 | | | | description: ""
    | | | Generate model
76 | | | - name: circularx_stock_possible_response
77 | | | | description: ""
    | | | Generate model
78 | | | - name: s3_easycash_ebay_catalog
79 | | | | description: ""
80 |

```

Annexe 6 :



Annex 7 :

```

1  import pandas as pd
2  import snowflake
3
4  def model(dbt, session):
5
6      ods_buyback_df = dbt.ref("ods_buyback")
7
8      return ods_buyback_df
9

```