YouCode / Youssoufia

Développeur Data

2023-2024

Projet : Evaluation (Sprint Talend)

YONLI Fidèle

Table des matières

Introduction

Ce projet vise à établir un entrepôt de données en utilisant SQL Server pour un agent de e-commerce. Il implique également l'utilisation de Talend pour l'ETL, en extrayant, transformant et chargeant les données dans l'entrepôt. Une constellation rapide de schémas est mise en place pour les ventes et l'inventaire, avec des tables de dimensions et de faits détaillées. Des DataMarts physiques pour les ventes et l'inventaire sont créés, suivis d'une analyse approfondie à l'aide de Power BI, comprenant la tendance des ventes, l'analyse des produits, la segmentation des clients, et plus encore. Le projet comprend également l'optimisation de DataMarts par l'indexation, la partition, l'agrégation, et d'autres techniques. Des politiques de rétention de données sont définies, la logique de transformation est validée, et des autorisations appropriées sont mises en place pour assurer la sécurité des données.

Gestion de projet

Pour mener à bien le projet, une organisation s'impose. Nous avons opté pour **Trello** comme plateforme de planification et d'organisation des différentes taches à réaliser. Ci-dessous une image de l'ensemble des taches réalisées et organisées.

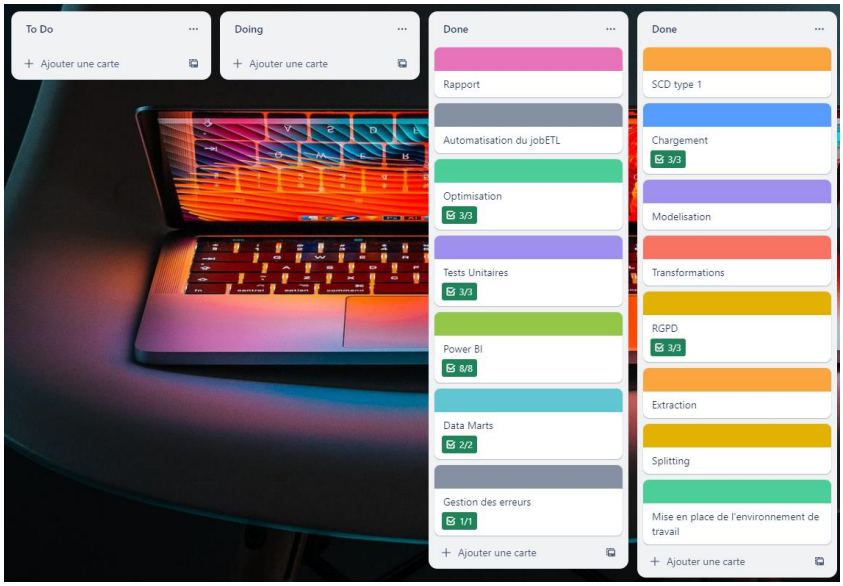


Figure 1: tableau trello

Processus ETL

Tout part d'un processus ETL (Extraction-Transformations-Load). Nous l'avons implémenté sous Talend Data Integration. Chaque étape a un ensemble de taches et d'objectifs précis.

Extraction

Cette étape constitue un job (**jExtraction**) à elle seule. Elle consiste à consolider (rassembler) les données issues des différentes sources en une vue unique. Pour ce faire, nous avons utilisé le composant **tUnite**. Puis, nous avons stocké les données consolidées dans une base de données (staging area) nommée **SprintTalend.** C'est dans cette zone de stockage que nous allons réaliser toutes les transformations nécessaires (de nettoyage principalement) pour construire le datawarehouse.

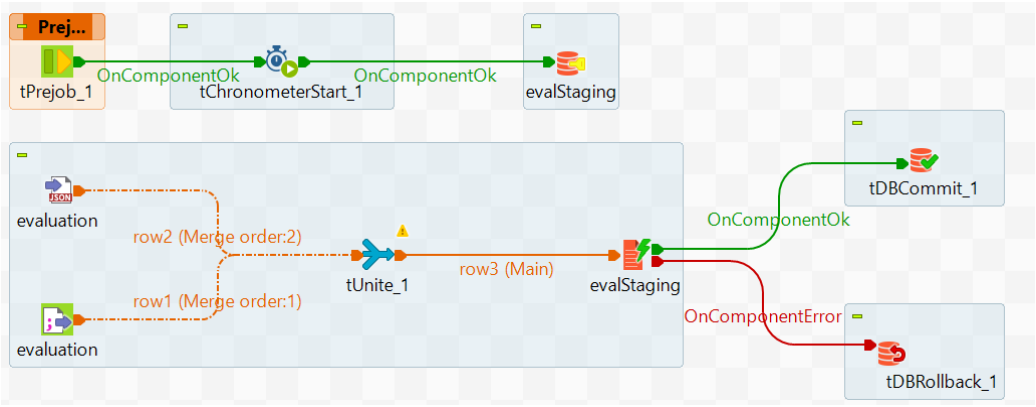


Figure 2: job jExtraction

Transformations

Cette étape consiste à faire subir à nos données le processus de nettoyage avant de les stocker dans notre datawarehouse. Plusieurs opérations ont été réalisées à cet effet sur plusieurs colonnes à l'aide du composant **tMap**:

- ✓ Uniformiser le format de Date *dd-MM-yyyy* dans la colonne **Date**
- ✓ Remplacer *InvalidPrice* par *TotalAmount/QuantitySold* dans la colonne **ProductPrice**
- ✓ Remplacer les valeurs manquantes de **CustomerEmail** et **SupplierContact**
- ✓ Crypter les colonnes **CustomerEmail**, **CustomerAddress**, **CustomerPhone**, **SupplierContact**

Puis, les données transformées sont transférées au job suivant via le composant **tBufferOutput**.

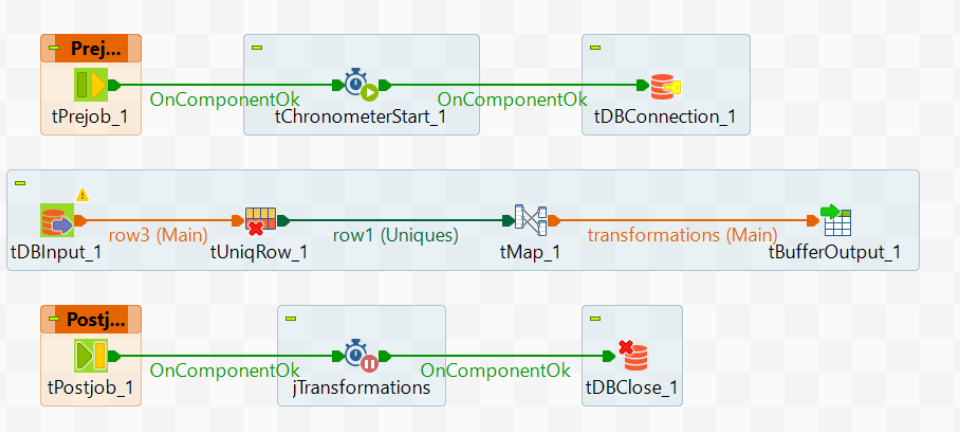


Figure 3: job jTransformarions

Modélisation du datawarehouse

Avant de passer au chargement dans notre entrepôt de données, nous avons besoin de préparer le datawarehouse. Nous allons donc modéliser notre entrepôt de données qui servira de lieu de chargement de nos données. Nous avons opté pour un schéma **fact constellation** pour plusieurs raisons :

- ✓ Plusieurs types d'évènements ou faits (inventaire, vente notamment) doivent être analysés ensemble
- ✓ Flexibilité d'analyse multidimensionnelle en croisant les données de différentes tables de faits
- ✓ Evolutif dans le sens où l'on peut ajouter d'autres évènements à notre entrepôt de données sans affecter le schéma déjà existant

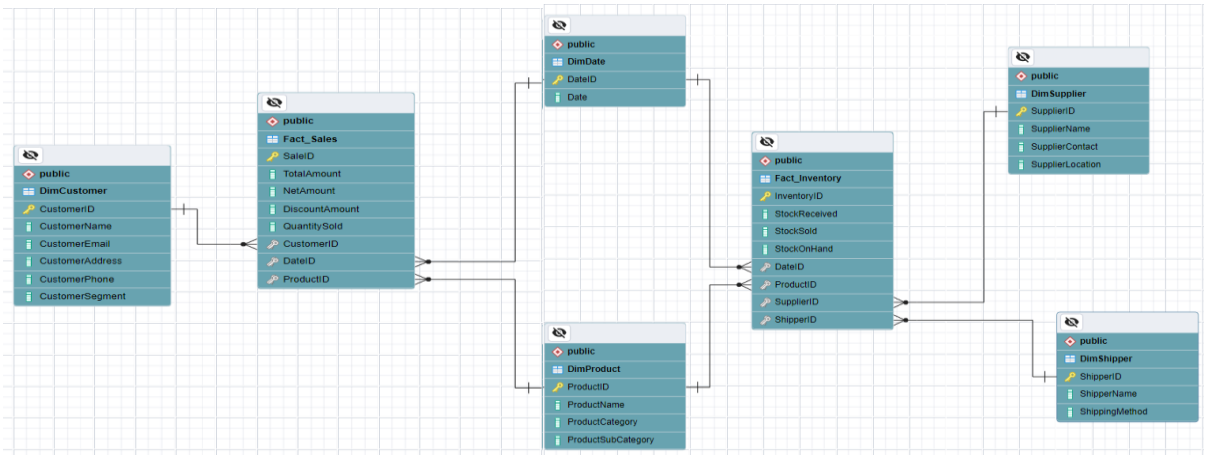


Figure 4: Modélisation du datawarehouse

Load (Chargement)

Tout est enfin pour que nous puissions peupler le datawarehouse des données transformées. Le chargement va s'effectuer en deux étapes :

- Chargement dans les tables de dimension
- Chargement dans les tables de fait

Chargement dans les tables de dimension

La création des dimensions se fait progressivement en s'appuyant sur notre modèle fourni. Nous avons principalement utilisé le composant **tMap** pour la sélection des colonnes par dimension. Ensuite, avec le composant **tUniqRow**, nous avons extrait les valeurs uniques. Puis, de nouveau à l'aide du composant **tMap**, nous générons et associons des identifiants à nos valeurs uniques pour enfin les stocker dans une table de notre datawarehouse avec le composant **tDBOutputBulkExec**.

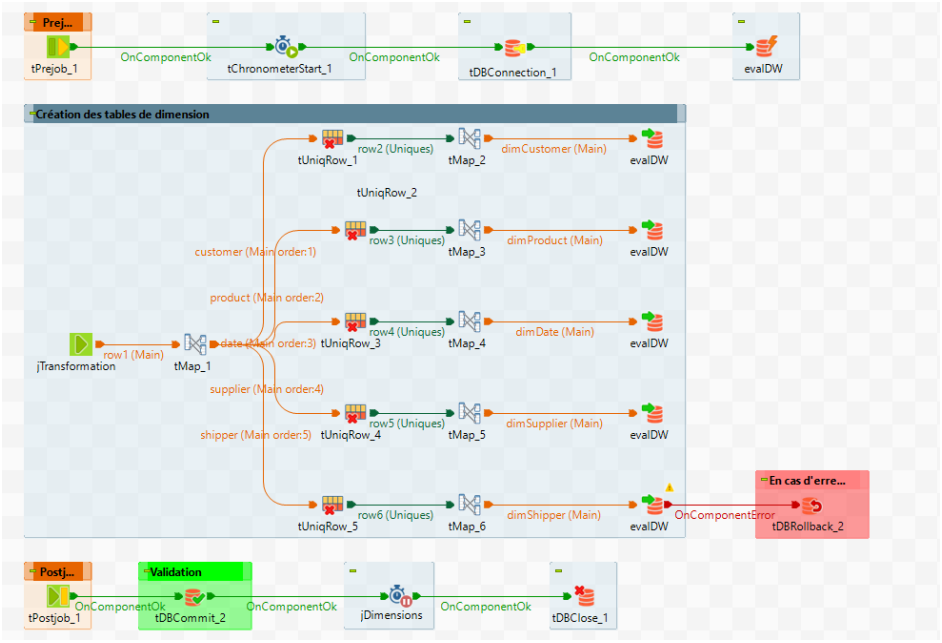


Figure 5: job jDimensions

Implémentation du SCD de type 1 (overwrite)

Pour implémenter le SCD, nous avions plusieurs options notamment avec le composant **Tdbscd**. Mais nous opté pour une méthode plus simple dans notre cas.

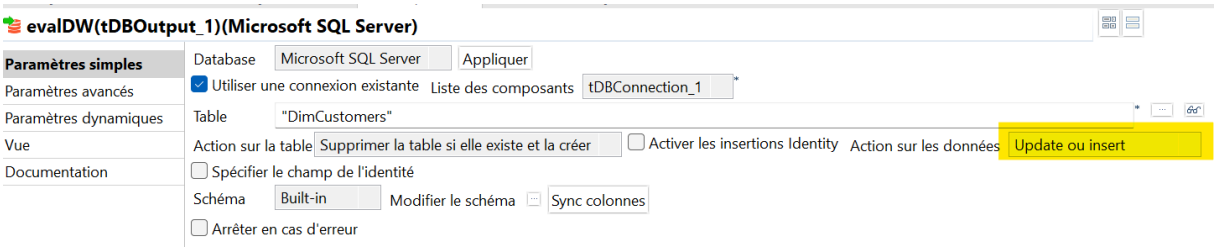


Figure 6: configuration de l'upsert (update or insert)

Les tables de dimensions sont créées. Nous allons pouvoir extraire leurs clés primaires pour les tables de faits.

Chargement dans les tables de fait

Les tables de fait sont les éléments de notre modèle qui contiennent les mesures et les identifiants des tables de dimension. Techniquement, pour leur construction, nous avons d'abord effectué des jointures entre les tables de dimension, puis avons remplacé les valeurs provenant des tables de dimension par leurs IDs pour constituer les tables de fait. De plus, nous conservons les données quantitatives nécessaires à notre analyse comme des colonnes de mesure. Tout ceci à l'aide du composant **tMap**.

Finalement, à l'aide du composant **tDBRow**, nous avons créé les clés étrangères pour établir les relations qui existent entre les tables selon notre modèle établi.

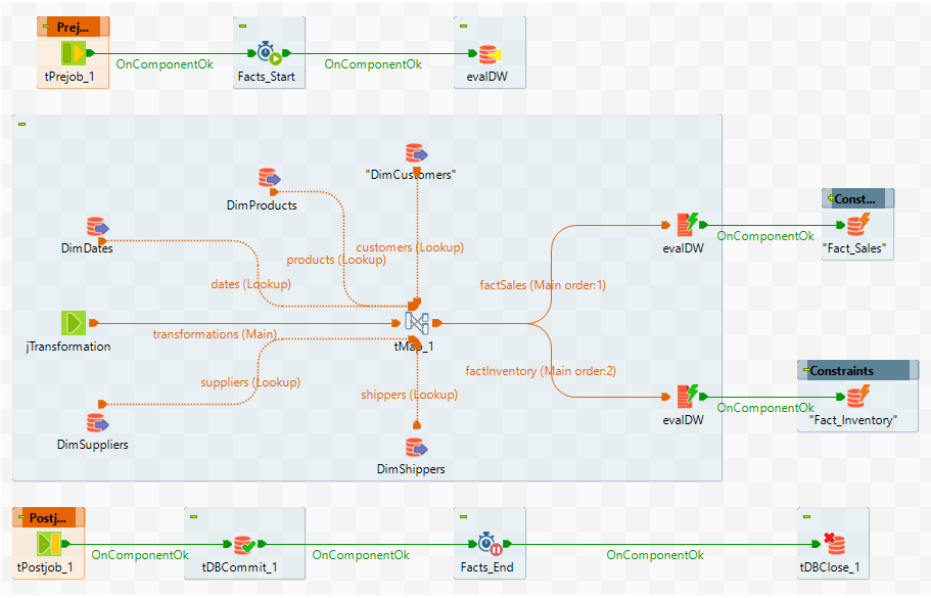


Figure 7: job jChargement

Tests Unitaires

Une fois les données chargées, il est important de procéder à une vérification pour nous assurer que les données sont conformes aux transformations, qu'elles ne sont pas altérées et qu'elles sont cohérentes. Pour cela, nous avons utilisé le framework **tSQLt** assisté de **dbForge**. La première chose consiste à installer le framewok.

```
Messages
   Installed at 2023-10-05 12:47:33.827

   +----------------------------------------+
   |                                        |
   | Thank you for using tSQLt.             |
   |                                        |
   | tSQLt Version: 1.0.8083.3529           |
   |                                        |
   +----------------------------------------+

   Completion time: 2023-10-05T12:49:26.4553301+02:00
```

Le processus de testing est le suivant :

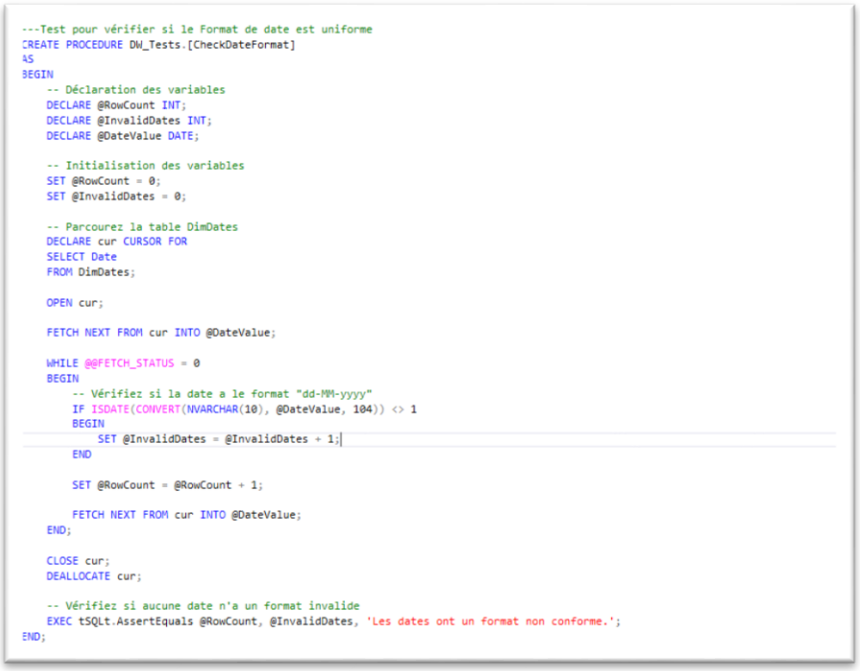
- Il faut tout d'abord créer une/plusieurs classe(s) qui contiendra(ont) les différents tests. Dans notre cas, nous avons créé une seule classe qui contiendra tous les tests de notre datawarehouse.

```
--Création d'une classe (schéma nommé)
EXEC tSQLt.NewTestClass 'DW_Tests';
```

- Ensuite, il faut créer les différents tests dans notre classe **DW_Tests**. Les tests réalisés sont les suivants :
  - ✓ Un test pour vérifier si les dates sont dans un format uniforme '***dd-MM-yyyy***'

```
---Test pour vérifier si le Format de date est uniforme
CREATE PROCEDURE DW_Tests.[CheckDateFormat]
AS
BEGIN
    -- Déclaration des variables
    DECLARE @RowCount INT;
    DECLARE @InvalidDates INT;
    DECLARE @DateValue DATE;

    -- Initialisation des variables
    SET @RowCount = 0;
    SET @InvalidDates = 0;

    -- Parcourez la table DimDates
    DECLARE cur CURSOR FOR
    SELECT Date
    FROM DimDates;

    OPEN cur;

    FETCH NEXT FROM cur INTO @DateValue;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Vérifiez si la date a le format "dd-MM-yyyy"
        IF ISDATE(CONVERT(NVARCHAR(10), @DateValue, 104)) <> 1
        BEGIN
            SET @InvalidDates = @InvalidDates + 1;
        END

        SET @RowCount = @RowCount + 1;

        FETCH NEXT FROM cur INTO @DateValue;
    END;

    CLOSE cur;
    DEALLOCATE cur;

    -- Vérifiez si aucune date n'a un format invalide
    EXEC tSQLt.AssertEquals @RowCount, @InvalidDates, 'Les dates ont un format non conforme.';
END;
```

✓ Un test pour vérifier que toutes les valeurs de la colonne ***TotalAmount*** sont positives



```
---Test pour vérifier si le Format de date est uniforme
CREATE PROCEDURE DW_Tests.[CheckDateFormat]
AS
BEGIN
    -- Déclaration des variables
    DECLARE @RowCount INT;
    DECLARE @InvalidDates INT;
    DECLARE @DateValue DATE;

    -- Initialisation des variables
    SET @RowCount = 0;
    SET @InvalidDates = 0;

    -- Parcourez la table DimDates
    DECLARE cur CURSOR FOR
    SELECT Date
    FROM DimDates;

    OPEN cur;

    FETCH NEXT FROM cur INTO @DateValue;

    WHILE @@FETCH_STATUS = 0
    BEGIN
        -- Vérifiez si la date a le format "dd-MM-yyyy"
        IF ISDATE(CONVERT(NVARCHAR(10), @DateValue, 104)) <> 1
        BEGIN
            SET @InvalidDates = @InvalidDates + 1;
        END

        SET @RowCount = @RowCount + 1;

        FETCH NEXT FROM cur INTO @DateValue;
    END;

    CLOSE cur;
    DEALLOCATE cur;

    -- Vérifiez si aucune date n'a un format invalide
    EXEC tSQLt.AssertEquals @RowCount, @InvalidDates, 'Les dates ont un format non conforme.';
END;
```

✓ Un test pour vérifier si la colonne ***CustomerEmail*** ne contient pas de valeurs manquantes

```
CREATE PROCEDURE DW_Tests.[TestMissingCustomerEmails]
AS
BEGIN
    -- Act (Action) : Sélectionnez toutes les valeurs de la colonne "Email" où le champ est NULL
    DECLARE @MissingEmails INT;

    SELECT @MissingEmails = COUNT(*)
    FROM DimCustomers
    WHERE CustomerEmail IS NULL;

    -- Assert (Vérification) : Vérifiez que le nombre de valeurs manquantes est égal à zéro
    EXEC tSQLt.AssertEquals 0, @MissingEmails;
END;
```

✓ Un test pour vérifier si la colonne ***SupplierContact*** ne contient pas de valeurs manquantes

```
CREATE PROCEDURE DW_Tests.[TestMissingCustomerEmails]
AS
BEGIN
    -- Act (Action) : Sélectionnez toutes les valeurs de la colonne "Email" où le champ est NULL
    DECLARE @MissingEmails INT;

    SELECT @MissingEmails = COUNT(*)
    FROM DimCustomers
    WHERE CustomerEmail IS NULL;

    -- Assert (Vérification) : Vérifiez que le nombre de valeurs manquantes est égal à zéro
    EXEC tSQLt.AssertEquals 0, @MissingEmails;
END;
```

- Enfin, nous exécutons nos tests pour nous assurer que tout est dans les normes souhaitées.

```
|No|Test Case Name                                |Dur(ms)|Result |
+--+---------------------------------------------+-------+-------+
|1 |[DW_Tests].[TestAllTotalAmountArePositive]|     60|Success|
|2 |[DW_Tests].[TestMissingCustomerEmails]    |    398|Success|
|3 |[DW_Tests].[TestMissingSupplierContacts]  |     18|Success|
-------------------------------------------------------------------------------------------
Test Case Summary: 3 test case(s) executed, 3 succeeded, 0 skipped, 0 failed, 0 errored.
-------------------------------------------------------------------------------------------

Completion time: 2023-10-08T15:05:00.7364582+02:00
```

Création des datamarts

Les données sont bien chargées dans les normes. Nous allons passer à la création de deux datamarts. L'un dédié à l'analyse des ventes et l'autre dédié à l'analyse des inventaires.

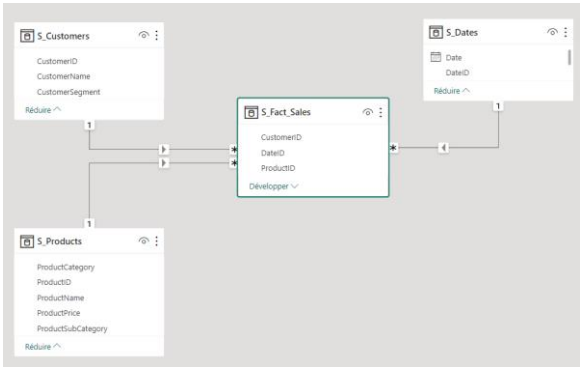


Figure 8 Modélisation du Sales Datamart

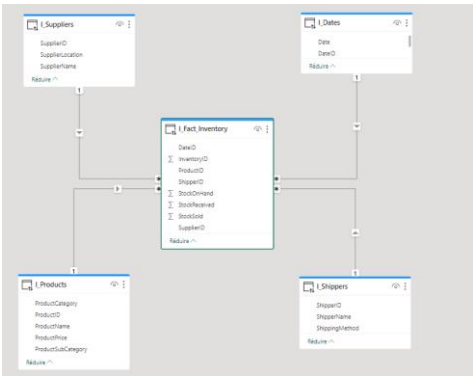


Figure 9: modélisation de Inventory Datamart

Pour cela, nous avons utilisé des requêtes SQL sous SQL Server chaque datamart constituant une base de donnée à part entière en deux étapes :

- Une première étape où l'on fait une copie des données du datawarehouse dans le datamart

```
CREATE DATABASE InventoryDataMart;

USE SprintTalendDW;

--Création des tables et copie des données dans les tables
--Tables de dimension
SELECT * INTO InventoryDataMart.dbo.I_Dates
FROM DimDates;

SELECT * INTO InventoryDataMart.dbo.I_Products
FROM DimProducts;

SELECT SupplierID, SupplierName, SupplierLocation INTO InventoryDataMart.dbo.I_Suppliers
FROM DimSuppliers;

SELECT * INTO InventoryDataMart.dbo.I_Shippers
FROM DimShippers;

--Table de fait
SELECT * INTO InventoryDataMart.dbo.I_Fact_Inventory
FROM Fact_Inventory;
```

Figure 10: Datamart SalesDataMart

```
CREATE DATABASE SalesDataMart;

USE SprintTalendDW;
------------------------------------------------------------------
--SalesDataMart
SELECT * INTO SalesDataMart.dbo.S_Dates
FROM DimDates;

SELECT * INTO SalesDataMart.dbo.S_Products
FROM DimProducts;

SELECT CustomerID, CustomerName, CustomerSegment
INTO SalesDataMart.dbo.S_Customers
FROM DimCustomers;

SELECT * INTO SalesDataMart.dbo.S_Fact_Sales
FROM Fact_Sales;
```

Figure 11: Copie du DW dans le DataMart InventoryDataMart

❖ Puis, une seconde étape où l'on crée les contraintes de clé primaire et de relation entre les différentes tables créées.

```
USE SalesDataMart;

--Définition des contraintes dans SalesDataMart
--Clés Primaires
ALTER TABLE S_Dates
ADD PRIMARY KEY (DateID);

ALTER TABLE S_Products
ADD PRIMARY KEY (ProductID);

ALTER TABLE S_Customers
ADD PRIMARY KEY (CustomerID);

ALTER TABLE S_Fact_Sales
ADD PRIMARY KEY (SaleID);

--Clés étrangères
ALTER TABLE S_Fact_Sales
ADD CONSTRAINT FK_SalesDate FOREIGN KEY (DateID) REFERENCES S_Dates(DateID);

ALTER TABLE S_Fact_Sales
ADD CONSTRAINT FK_SalesCustomer FOREIGN KEY (CustomerID) REFERENCES S_Customers(CustomerID);

ALTER TABLE S_Fact_Sales
ADD CONSTRAINT FK_SalesProduct FOREIGN KEY (ProductID) REFERENCES S_Products(ProductID);
```

Figure 12: Définition des contraintes dans le datamart SalesDataMart

```
USE InventoryDataMart;

--Définition des contraintes
--Clés Primaires
ALTER TABLE I_Dates
ADD PRIMARY KEY (DateID);

ALTER TABLE I_Products
ADD PRIMARY KEY (ProductID);

ALTER TABLE I_Suppliers
ADD PRIMARY KEY (SupplierID);

ALTER TABLE I_Shippers
ADD PRIMARY KEY (ShipperID);

ALTER TABLE I_Fact_Inventory
ADD PRIMARY KEY (InventoryID);

--Clés étrangères
ALTER TABLE I_Fact_Inventory
ADD CONSTRAINT FK_InventoryDate FOREIGN KEY (DateID) REFERENCES I_Dates(DateID);

ALTER TABLE I_Fact_Inventory
ADD CONSTRAINT FK_InventoryProduct FOREIGN KEY(ProductID) REFERENCES I_Products (ProductID);

ALTER TABLE I_Fact_Inventory
ADD CONSTRAINT FK_InventorySpplier FOREIGN KEY (SupplierID) REFERENCES I_Suppliers (SupplierID);

ALTER TABLE I_Fact_Inventory
ADD CONSTRAINT FK_InventoryShipper FOREIGN KEY (ShipperID) REFERENCES I_Shippers (ShipperID);
```

Figure 13: Définition des contraintes dans le datamart InventoryDataMart

RGPD & Optimisations & Automatisation

Le respect du RGPD

Le respect du RGPD est essentiel dans un projet data comme le notre et en tant que data developer, il est de notre ressort et de notre devoir de le respect. Pour cela, les colonnes ***CustomerEmail***, ***CustomerAddress***, ***CustomerPhone***, ***SupplierContact*** ont été crypté à défaut de les supprimer pour protéger les protéger.

Optimisations

Notre attention s'est portée également sur l'optimisation de nos infrastructures. Nous avons principalement appliquer les techniques **l'indexation** et le **partitionnement** uniquement sur le datawarehouse. Nous ne les avons pas appliquées sur les datamarts principalement parce que nous n'effectuons pas de requêtes SQL sur nos datamarts. Ils sont dédiés dans notre cas à la réalisation de visualisation sur PowerBI.

Automatisation du job ETL

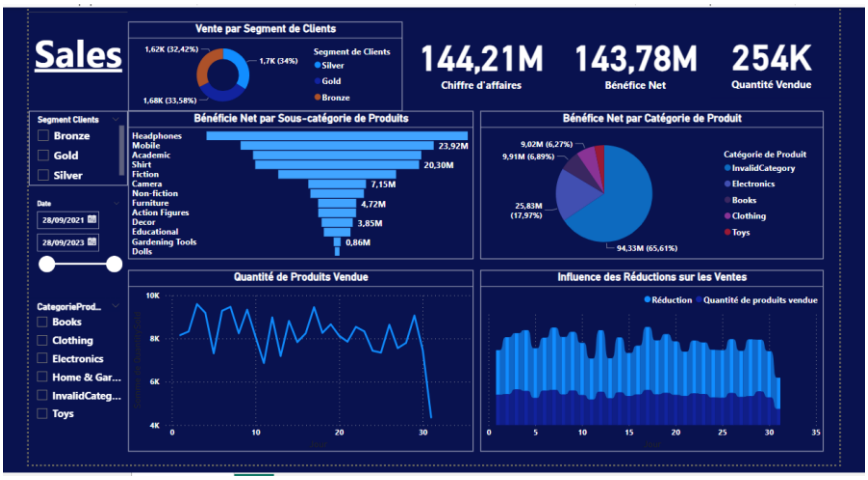
Nous avons automatiser l'exécution du job ETL pour qu'il s'exécute chaque jour à 02h.

Pour cela, nous avons exporter le job principal sous format .bat et avons planifié son exécution avec le planificateur de taches de Windows.

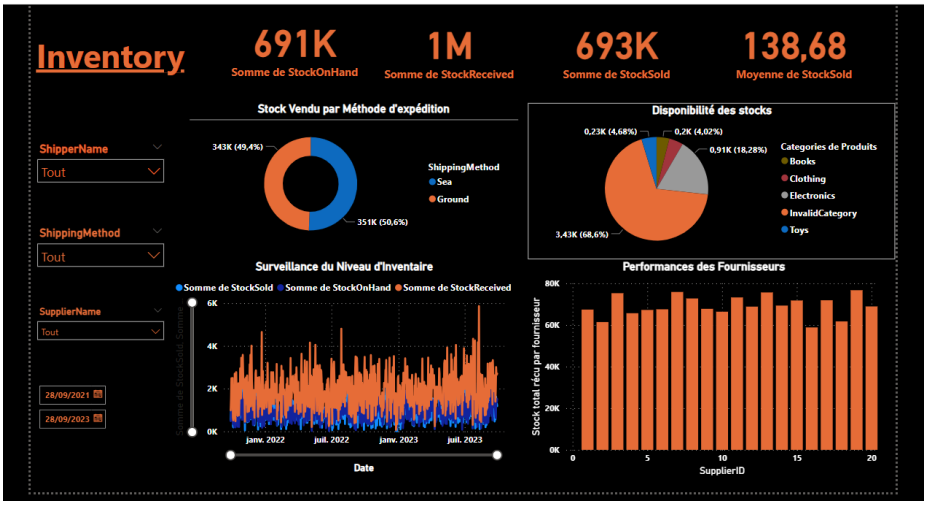
Visualisations avec PowerBI

Pour nous visualisations, nous avons opté pour **PowerBi** pour sa simplicité et la facilité à créer des visuels. C'est l'outil idéal pour créer rapidement des tableaux de bord bien fournis pour nos analyses. Nous avons créé deux tableaux de bord, chacun dédié à l'analyse d'un datamart.

- ## Le tableau de bord des ventes avec des données importées



- ## Le tableau de bord des inventaires par direct Query



Conclusion

A la fin de ce projet, nous sommes fiers de l'avoir réalisé. En tant que data engineer, nous avons mis un accent particulier sur le respect du RGPD avec le cryptage des données sensibles, et les optimisations (l'indexation et le partitionnement) et des bonnes pratiques.