YouCode / Youssoufia

Développeur Data

2023-2024

Projet : Optimisation de l'Entrepôt de Données pour un Acteur Majeur de l'E-Commerce

SQUAD 2

Yassine ESSADI : Data Engineer

Fidèle YONLI : Scrum Master

Hicham TARIFI : Data Engineer

Yassine HARRATI : Data Engineer

Table des matières

Introduction

Ce projet se concentre sur la création d'un entrepôt de données pour une plateforme e-commerce, couvrant principalement les ventes et l'inventaire. L'accent est mis sur la modélisation des données, la transformation ETL, l'optimisation SQL, la conception de cubes OLAP et les tests unitaires SQL.

XYZ Corp, l'une des plus grandes entreprises de e-commerce en France, se trouve face à un défi croissant de gestion et d'analyse de ses données. Avec une base croissante de clients et un catalogue de produits qui s'élargit chaque jour, l'entreprise ressent le besoin d'une solution robuste pour analyser efficacement ses ventes et gérer son inventaire.

En tant que développeur Data, notre mission sera centrée sur l'implémentation d'un ETL optimisé à l'aide de talend, la mise en œuvre d'un entrepôt de données optimisés, voici les principales tâches qui nous sont confiées :

Définition des processus de mise en conformité avec le RGPD : Élaborer un plan pour s'assurer que toutes les données sont traitées conformément au RGPD.

- **Implémentation d'un Job ETL optimisé et la gestion des erreurs du job**
- **Automatisation du Job ETL à l'aide Gestionnaire des tâches**
- **Modélisation et Optimization de l'entrepôt de données** : L'implémentation du schéma fact constellation et Partitionné sur Date ID pour de meilleures performances de requête sur les analyses basées sur le temps.
- **La création de cubes OLAP et l'exécution de requêtes MDX pour une analyse rapide** : Exemples de requêtes : Ventes totales par catégorie de produits et mois, Niveaux de stock pour un produit spécifique dans les entrepôts.
- **Tests Unitaires SQL** : Implémenter des tests pour s'assurer de la qualité et de la fiabilité des données et des transformations. Par exemple, Assurez-vous qu'aucune valeur de **TotalAmount** n'est négative grâce l'implémentation d'une procédure SQL.

Nous avons donc organisé le travail en plusieurs étapes et taches. Nous commençons par le processus d'Extraction, Transformation et Chargement (**ETL**).

Gestion de projet

Pour mener à bien le projet et le gérer efficacement, un système de répartition de taches a été appliqué en vue de combiner et converger les efforts de chaque membre de l'équipe vers un objectif précis. Le diagramme de GANTT suivant présente les détails de l'assignation des taches individuelles.

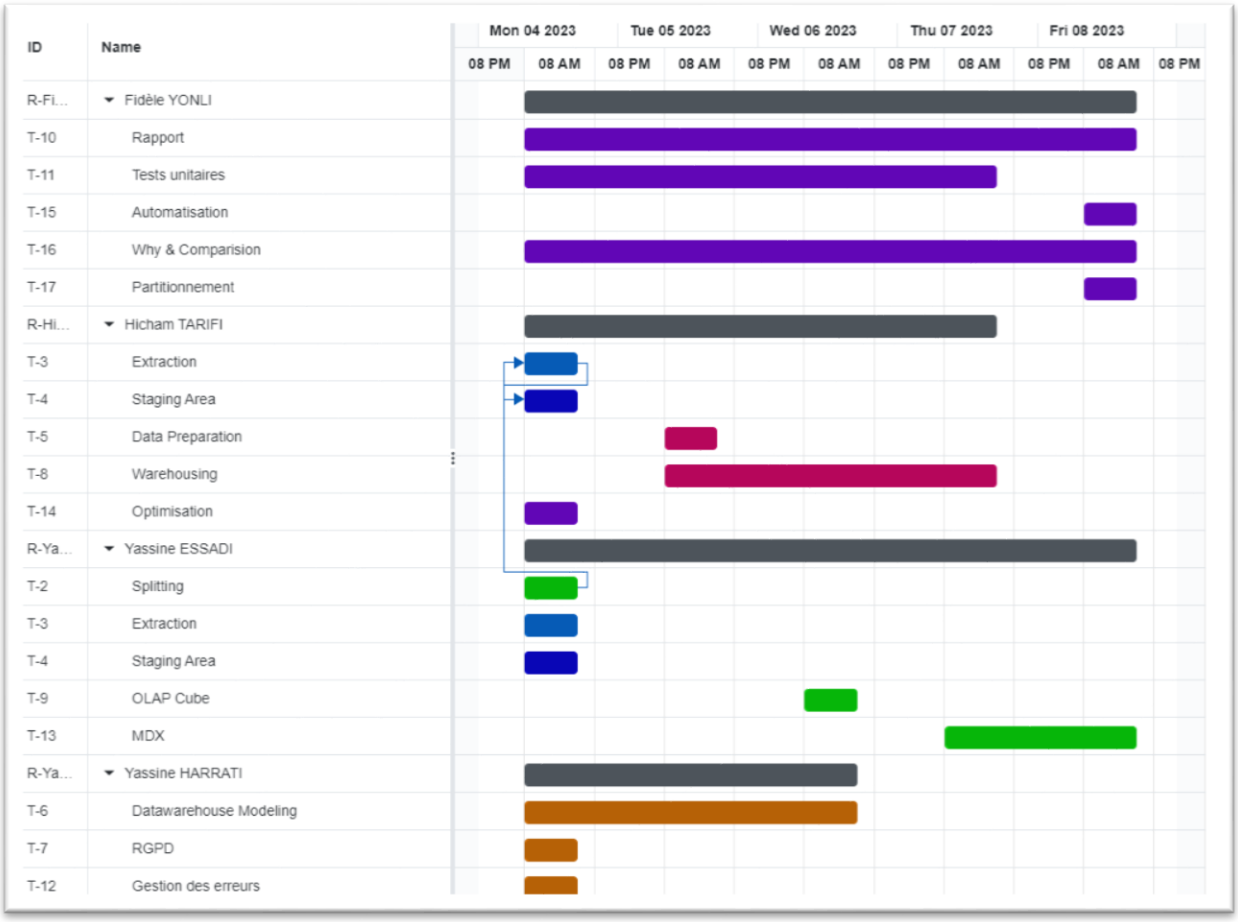


Figure 1 : Diagramme de GANTT

Les taches réparties, nous nous sommes attelés à les réaliser. Les lignes suivantes explicitent ce quoi a été concrètement effectué à chaque étape de la réalisation du projet.

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 **Sales_Olap_Staging_Area_DB.** C'est dans cette zone de stockage que nous allons réaliser toutes les transformations nécessaires (de nettoyage principalement) pour construire le datawarehouse.

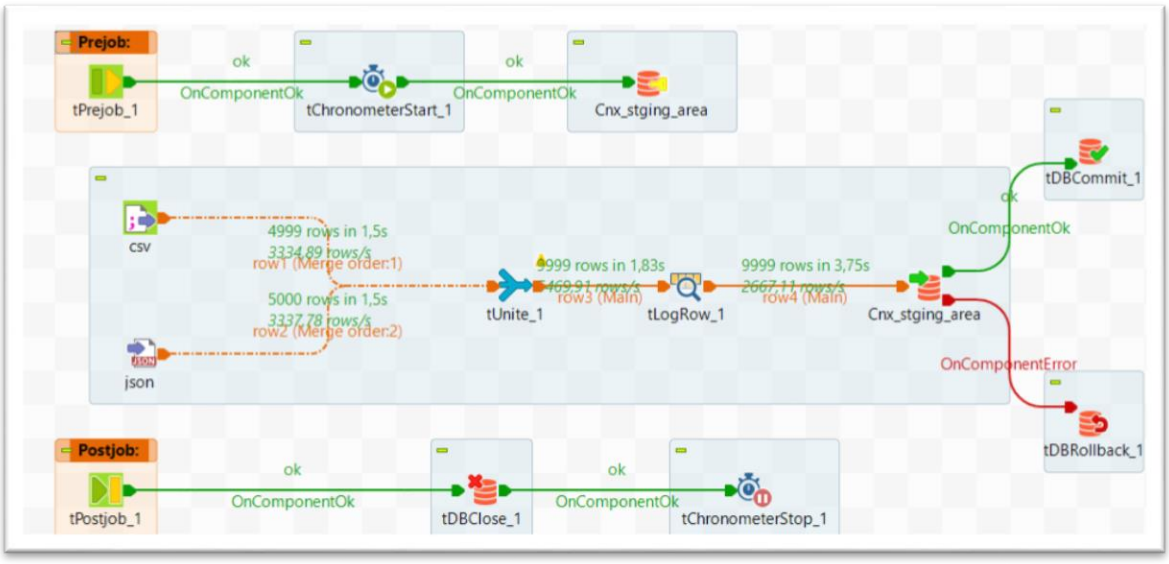


Figure 2: jExtraction

Transformation

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**:

- Colonne **Email** : remplacer les valeurs manquantes par « email not found » parce qu'il n'y a pas moyen de trouver l'email exact.
- Colonne **Age** : utiliser la valeur absolue des âges négatifs, remplacer les valeurs supérieures à 100 par la médiane mieux adaptée ici car elle n'est pas sensible à
- Colonne **Category** : supprimer les caractères spéciaux
- Colonne **Date_Order** : uniformiser les formats
- Colonne **TotalAmount** : uniformiser les formats
- Colonne **QualityScore** : remplacer les valeurs manquantes par 0
- Colonne **LastRestockDate** : garder le même format que celui de **Date_Order**

Cette étape est réalisée dans un job nommé **jSales** comme présenté dans la figure sous-dessous.

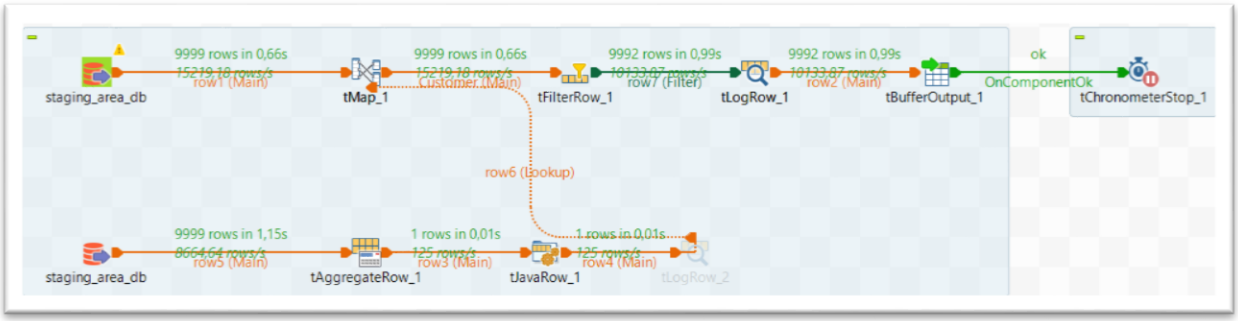


Figure 3: jSales pour les transformations:

Datawarehouse modeling

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

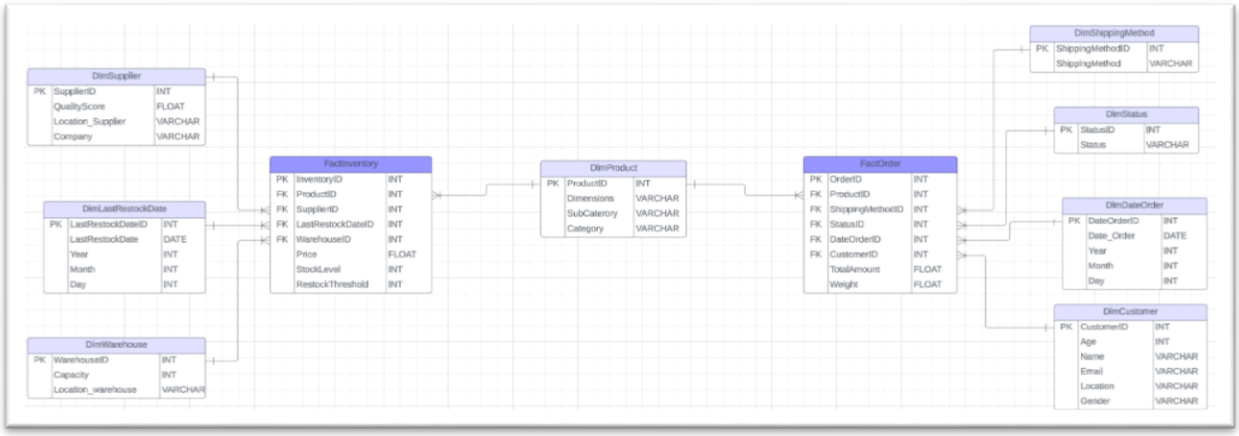


Figure 4: schéma fact constellation du datawarehouse

Création des tables de dimension

La création des dimensions se fait progressivement en s'appuyant sur notre modèle fourni allant des tables-feuilles vers le les tables-troncs en considérant que notre modèle est un arbre et que les sous-dimensions sont des feuilles, les dimensions les branches et les fact tables le tronc.

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 **tDBOutput**.

Les figures ci-dessous illustrent tout le processus réparti en différents jobs.

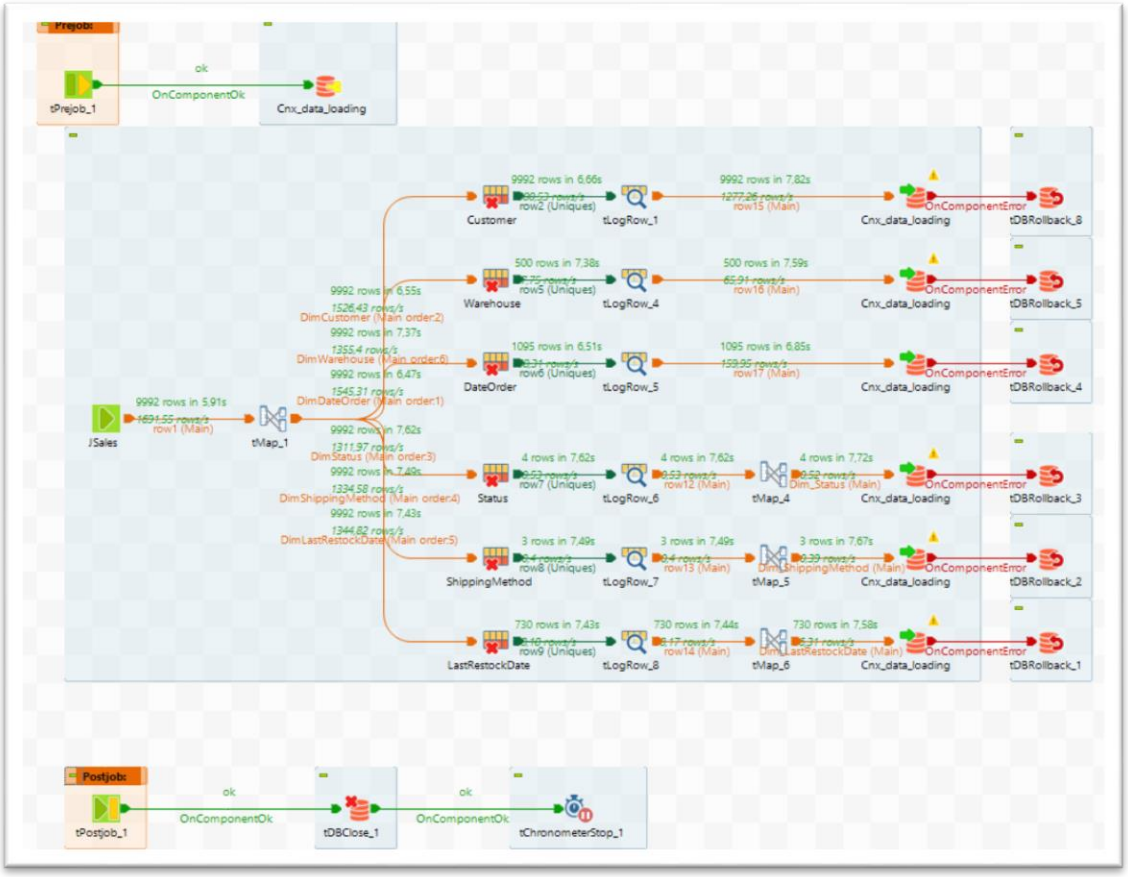


Figure 5: dimensions

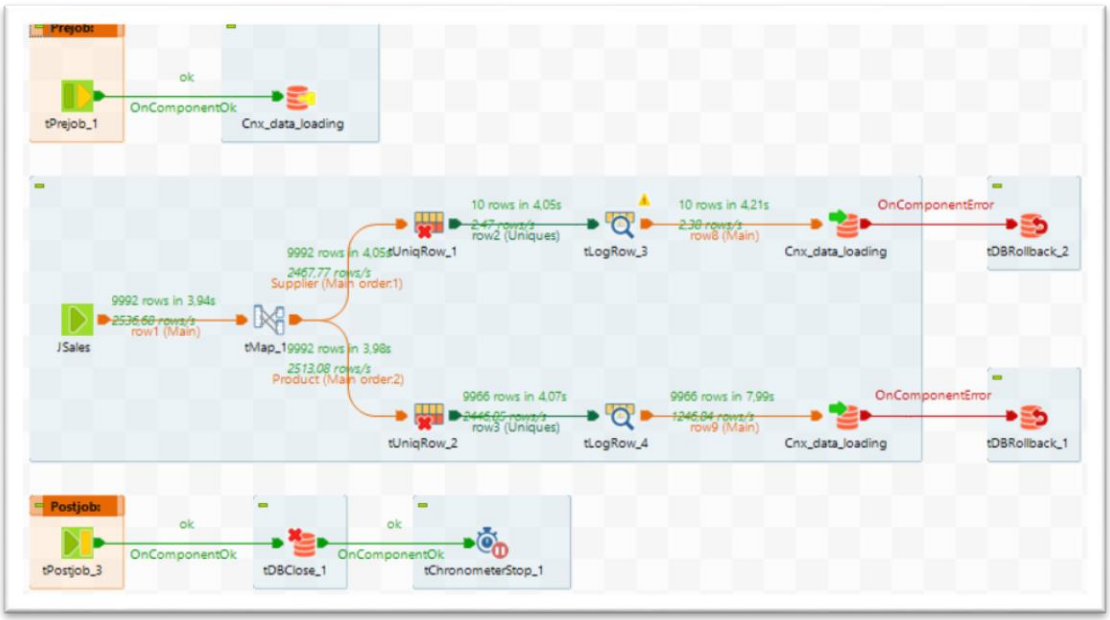


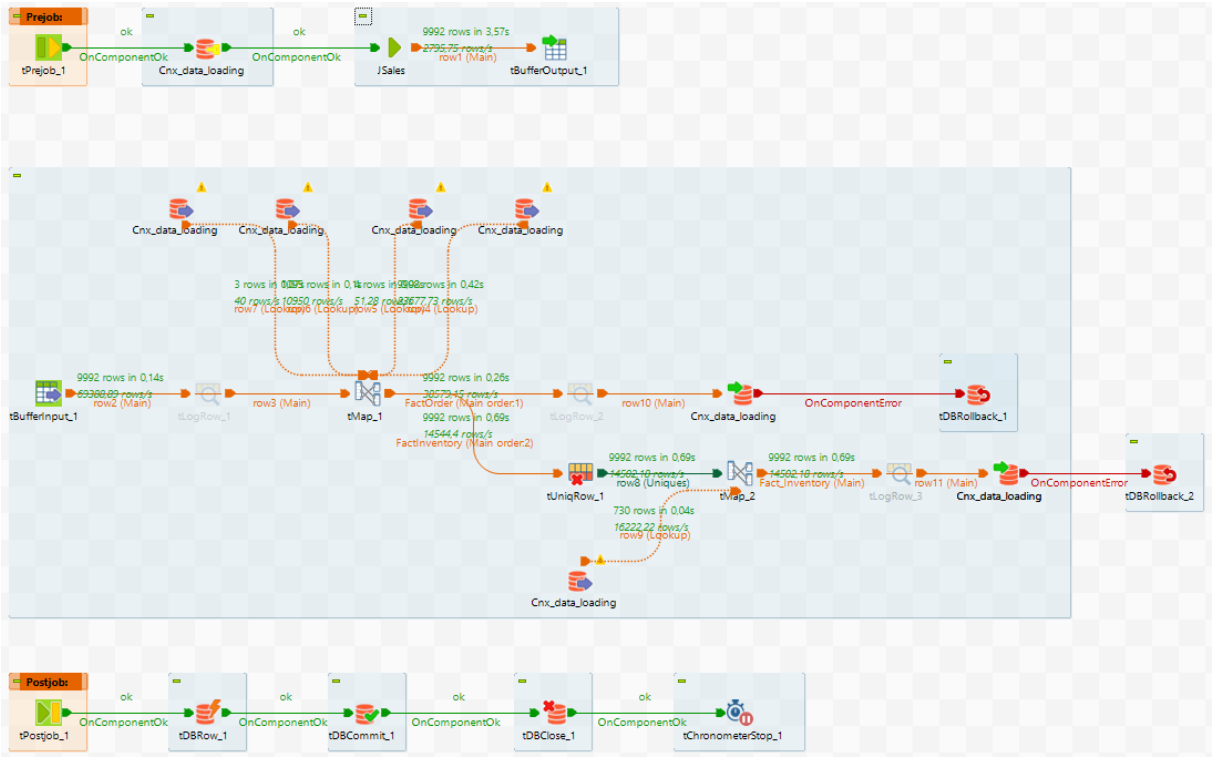
Figure 6: dimensions Supplier et Product

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

Création des 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.

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.



Tests Unitaires

Pour nous assurer que les données sont conformes à nos besoins, nous allons effectuer une série de tests unitaires.

Dans le contexte d'une base de données, un test unitaire fait référence à une pratique visant à tester individuellement les composants de la base de données de manière isolée et automatisée. L'objectif principal est de s'assurer que chaque composant fonctionne correctement, qu'il produit les résultats attendus, qu'il répond aux spécifications de manière cohérente.

Dans notre cas, nous avons effectué 09 tests unitaires pour nous assurer justement que nos données transformées respectent les normes de transformations voulues.

Ad hoc, nous avons opté pour le Framework de tests unitaires **tSQLt** dédié spécialement à SQL server. Il fonctionne sur la base de procédures qui contiennent les tests à réaliser. Les tests sont organisés en schéma qui ne sont rien d'autre que des classes. Autrement dit, les tests

sont les objets des schémas. TSQLt facilite les tests unitaires en ce sens qu'il possède déjà des procédures stockées qui accélèrent l'implémentation des tests.

Pour utiliser **tSQLt**, il faut d'abord l'installer dans la base de données. En ce qui nous concerne, nous l'avons installé dans notre datawarehouse.

Tous nos tests sont contenus dans le schéma suivant que nous avons créé :

```
----Création du schéma
EXEC tSQLt.NewTestClass 'Sales_Olap_Loading_DW_Tests';
```

A présent, nous voulons vérifier que toutes les valeurs de la colonne **TotalAmount** sont positives. Nous allons créer un test à cet effet dans la classe **Sales_Olap_Loading_DW_Tests**.

```
--TestTotalAmount
CREATE PROCEDURE Sales_Olap_Loading_DW_Tests.[TestTotalAmount]
AS
BEGIN
    -- Act (Action) : Vérifiez si toutes les valeurs de la colonne "TotalAmount" sont positives
    DECLARE @NegativeTotalAmount INT;

    SELECT @NegativeTotalAmount = COUNT(*)
    FROM FactOrder
    WHERE TotalAmount < 0;

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

Puis nous l'exécutons avec cette commande.

```
EXEC tSQLt.Run 'Sales_Olap_Loading_DW_Tests.[TestTotalAmountTestTotalAmount]' ;
```

Nous procédons de la même façon pour l'ensemble des tests en prenant soin de personnaliser chaque test.

L'exécution de tous les tests est possible grâce à une commande unique.

```
--Exécution de tous les tests
EXEC tSQLt.RunAll;
```

```
|No|Test Case Name                                                |Dur(ms)|Result |
+--+--------------------------------------------------------------+-------+-------+
|1 |[Sales_Olap_Loading_DW_Tests].[testAgesOver13]                |      0|Success|
|2 |[Sales_Olap_Loading_DW_Tests].[TestCapacity]                  |     48|Success|
|3 |[Sales_Olap_Loading_DW_Tests].[TestMissingEmails]             |      0|Success|
|4 |[Sales_Olap_Loading_DW_Tests].[TestMissingQualityScore]       |     16|Success|
|5 |[Sales_Olap_Loading_DW_Tests].[TestPrice]                     |      0|Success|
|6 |[Sales_Olap_Loading_DW_Tests].[TestRestockThreshold]          |      0|Success|
|7 |[Sales_Olap_Loading_DW_Tests].[TestStockLevel]                |     16|Success|
|8 |[Sales_Olap_Loading_DW_Tests].[TestTotalAmount]               |     16|Success|
|9 |[Sales_Olap_Loading_DW_Tests].[TestWeight]                    |      0|Success|
-------------------------------------------------------------------------------------
Test Case Summary: 9 test case(s) executed, 9 succeeded, 0 skipped, 0 failed, 0 errored.
-------------------------------------------------------------------------------------

Completion time: 2023-09-07T16:59:45.4664100+02:00
```

Figure 7: Résultat des tests unitaires

Gestion des erreurs-Optimisations-Automatisation-RGPD

Gestion des erreurs

Gérer les erreurs offre plusieurs avantages :

- Prévention des défaillances catastrophiques
- Maintenabilité
- Meilleure expérience utilisateur
- Sécurité

Pour toutes ces raisons, nous avons pris en compte la gestion des erreurs dans notre projet.

Talend nous offre une large gamme de composants pour gérer les erreurs. Nous avons utilisé principalement le composant suivant : **TDBRollback.**

Comment optimiser le datawarehouse ?

L'optimisation des performances est toujours l'un des points essentiels sur lesquels nous mettons la lumière. C'est pourquoi nous avons pensé aux performances tout au long de notre projet.

- **Performances au niveau du job ETL**
  - Respect des bonnes pratiques
  - Passage de données d'un job à un autre
  - Job pour chaque étape importante
  - Choix des composants adaptés
- **Performances au niveau du datawarehouse**
  - Choix du schéma convenable
  - Implémentation du partitionnement sur les tables volumineuses
  - Création des index
  - Respect des bonnes pratiques

En ce qui concerne le partitionnement, il y a un processus à suivre :

- D’abord, créer les fonctions de partition
- Ensuite, créer les schémas de partition
- Enfin, créer les tables de partition

```
--Création de fonctions de partitionnement
--Fact_Inventory
CREATE PARTITION FUNCTION PF_InventoryDate (DATE)
AS RANGE LEFT FOR VALUES
    ('2023-01-01','2023-03-01','2023-06-01', '2023-09-01', '2023-12-01');

--Fact_Order
-- Créez une fonction de partitionnement pour Fact_Order
CREATE PARTITION FUNCTION PF_OrderDate (DATE)
AS RANGE LEFT FOR VALUES
    ('2023-01-01','2023-03-01', '2023-06-01','2023-09-01', '2023-12-01');
```

```
--Création des schema de partition
--Fact_Inventory
-- Créez un schéma de partition pour Fact_Inventory
CREATE PARTITION SCHEME PS_InventoryDate
AS PARTITION PF_InventoryDate
TO ([PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY]);


--Fact_Order
-- Créez un schéma de partition pour Fact_Order
CREATE PARTITION SCHEME PS_OrderDate
AS PARTITION PF_OrderDate
TO ([PRIMARY], [PRIMARY], [PRIMARY], [PRIMARY]);
```

```
-- Créez la table Fact_Inventory partitionnée
CREATE TABLE Fact_Inventory
(
    InventoryID INT PRIMARY KEY,
    Date DATE,
    ProductID INT,
    Quantity INT
)
ON PS_InventoryDate(Date);


--Fact_Order
-- Créez la table Fact_Order partitionnée
CREATE TABLE Fact_Order
(
    OrderID INT PRIMARY KEY,
    Date DATE,
    CustomerID INT,
    TotalAmount DECIMAL(18, 2)
)
ON PS_OrderDate(Date);
```

Automatisation du job ETL

Pour l’automatisation, vu que nous utilisons la solution ETL Talend, nous aurions pu utiliser **Talend Administration Center** mais c’est une option payante. Donc, nous allons utiliser le **planificateur de taches de Windows**, notre système d’exploitation.

Pour ce fait, nous allons construire le job à automatiser c'est-à-dire que nous allons former un fichier **.bat** à exécuter à base du job général qui contient tous les sous-jobs qui constituent toutes les étapes du processus **ETL**.

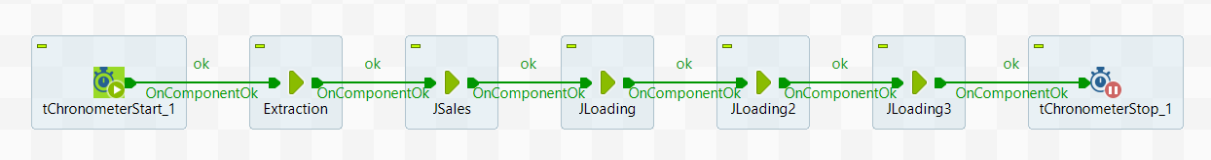
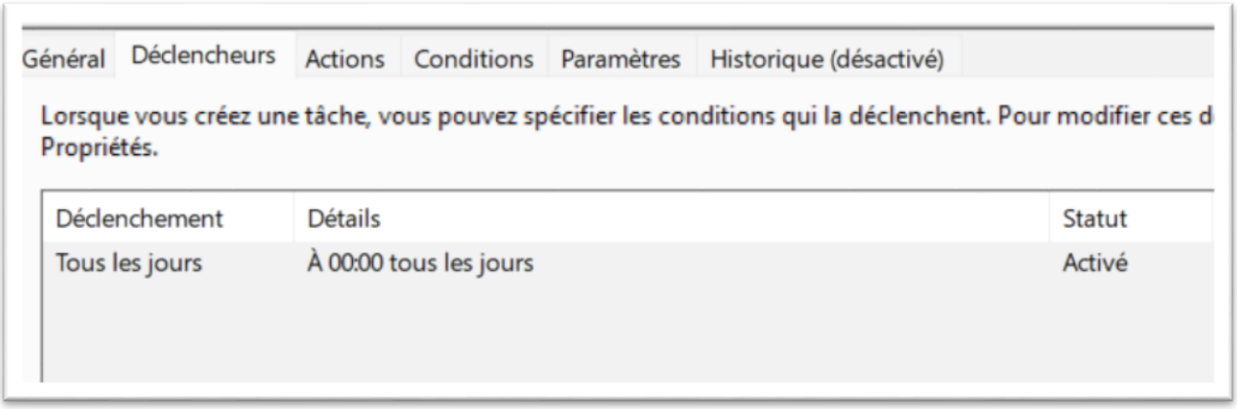
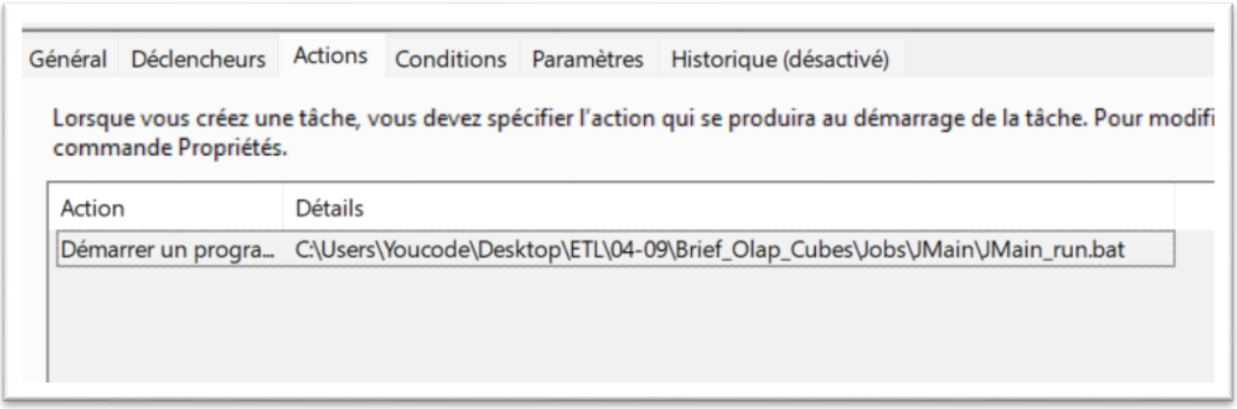


Figure 8:job principal

Puis, à l'aide du planificateur de taches, nous allons paramétrer l'exécution selon nos conditions et besoins ;





Ainsi, chaque jour à **00:00:00**, le job ETL s'exécute.

Le RGPD est-il respecté ?

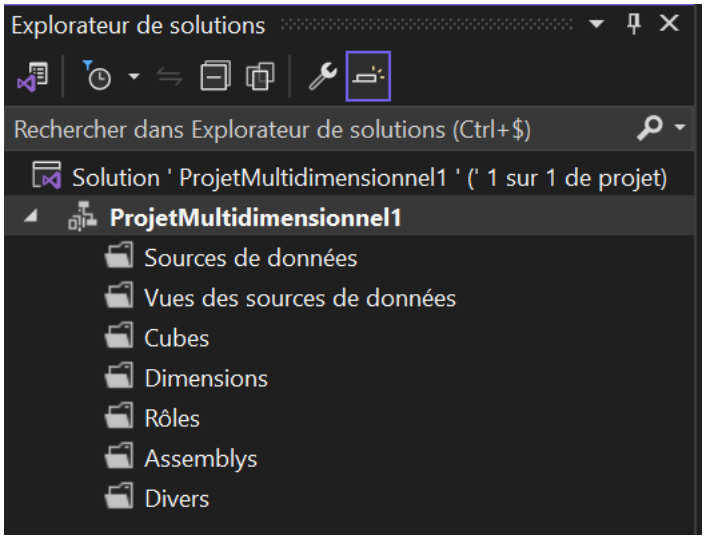
En tant qu'équipe de développeurs data, le respect des données personnelles et sensibles est une priorité. C'est pourquoi, nous avons voulu rester rigoureux quant au respect des normes du **RGPD** (Règlement Général de Protection des Données personnelles).

Concrètement, nous avons encrypté les données de la colonne **email** qui est une information sensible. De plus, avoir le mail ne nous servira pas dans notre analyse. Donc quitte à compromettre leur sécurité (une fuite par exemple), autant les encrypter. De plus, nous supprimé les **Ages <13** pour la protection des mineures selon le RPGD.

Création du cube OLAP

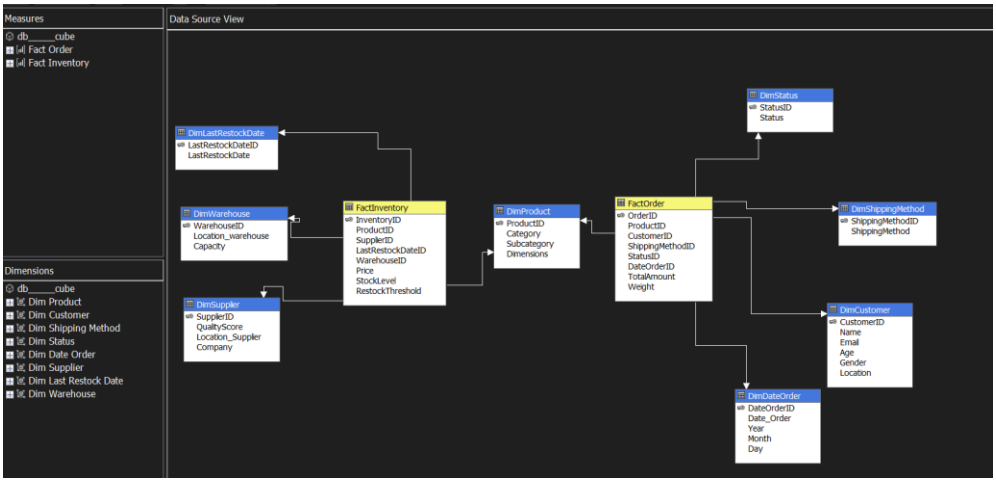
A présent, nous allons construire un cube OLAP multidimensionnel pour faciliter l'analyse. Mais pour cela, nous avons besoin d'un server MOLAP comme SSAS (SQL Server Analysis Services). Une fois installé et configuré, nous sommes passés au processus de création.

Sous Visual Studio 2022, nous créons un nouveau Projet Multidimensionnel Analysis Services. Puis, nous passons à l'importation des données de notre datawarehouse.



Pour chaque section :

- ✓ **Sources de données** : nous effectuons une connexion avec notre datawarehouse
- ✓ **Vues des sources de données** : nous visualisons le schéma du datawarehouse

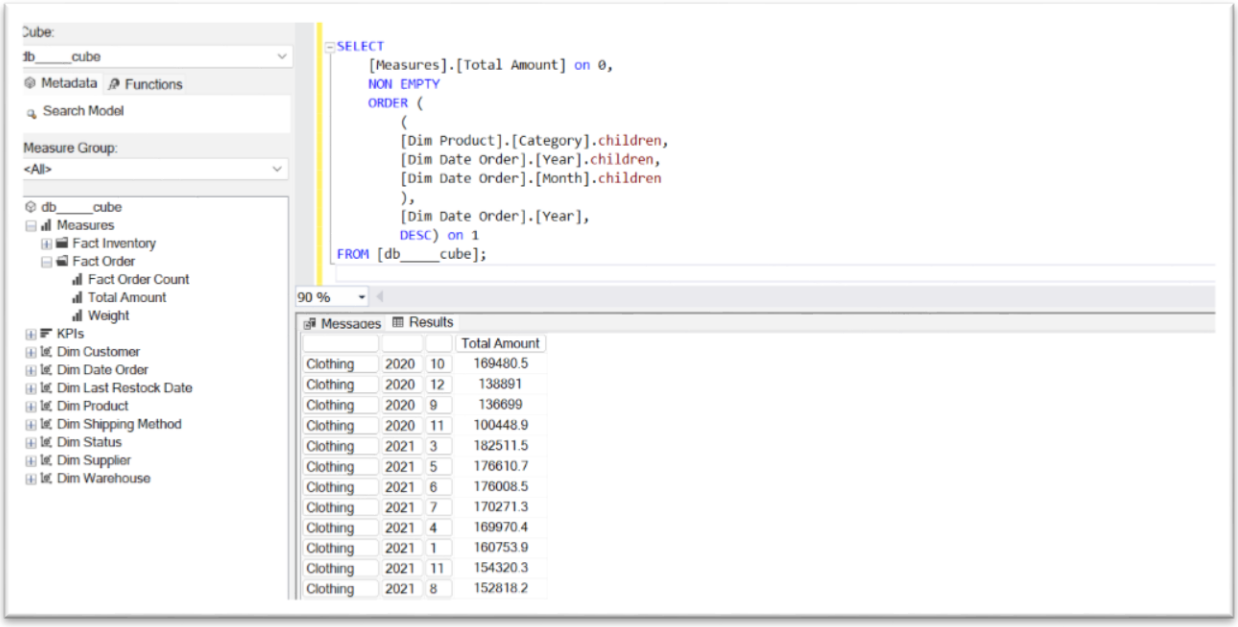


- ✓ **Cubes** : nous importons les tables de fait et les relations entre elles.

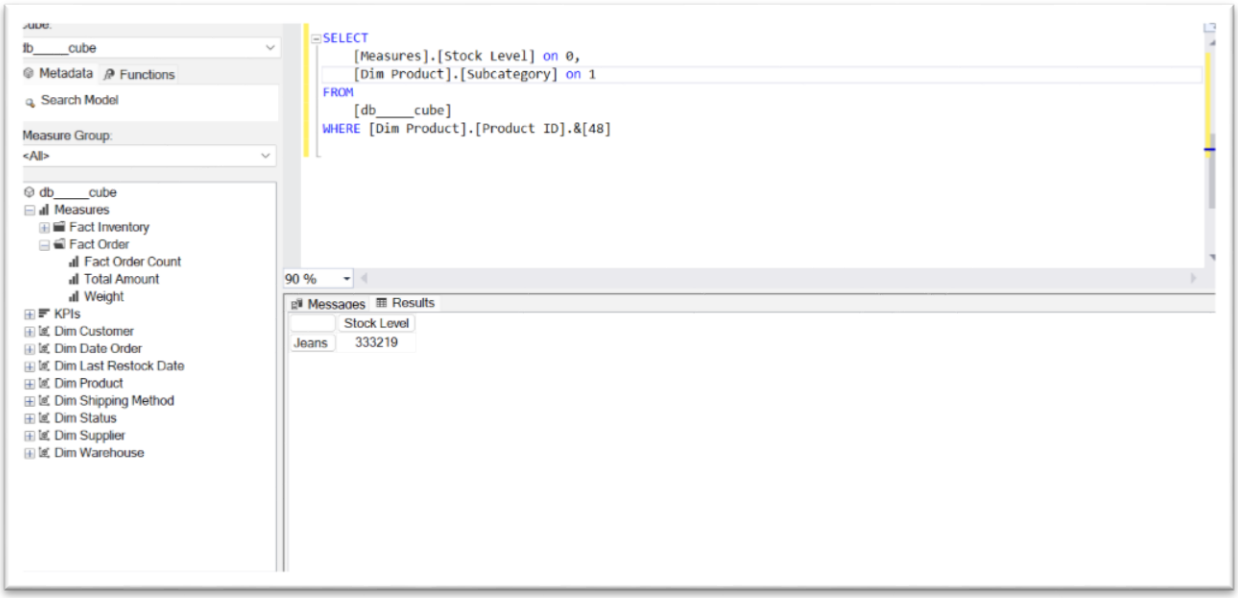
Requêtes MDX

Grace au cube OLAP que nous avons construit (build) et déployer (deploy), nous effectuons des requêtes MDX, langage pour des requêtes destinées à des analyses multidimensionnelles.

Par exemple, les ***Ventes totales par catégorie de produits et mois :***



Un autre exemple, les ***Niveaux de stock pour un produit spécifique dans les entrepôts***



Puis, nous nous sommes amusés à créer les mêmes requêtes en SQL. Nous avons obtenu bien sur les mêmes résultats. Cependant, nous constatons une différence de performance de requête. Le MDX est plus optimisé et plus convenable dans ce contexte.

Conclusion

En somme, ce projet malgré son lot de défis, a été mené à bon port. Les objectifs sont atteints dans le respect du cahier de charges tout en veillant au respect et à la protection des données sensibles et personnelles.