



DW and BI

Projet 1 : Création d'un Data Warehouse

Réalisé par :

BELHOUCHA Mohamed

Encadré par :

Prof. BOUAIN Abelhadi

Le sommaire

1. Introduction.....	3
1.1 Contexte du projet.....	3
1.2 Objectifs du projet.....	3
1.3 Périmètre fonctionnel.....	4
1.4 Architecture décisionnelle mise en place.....	4
2. Analyse des besoins décisionnels.....	5
2.1. Problématique métier.....	5
2.2. Questions décisionnelles principales.....	5
2.3. Mesures identifiées.....	6
2.4. Dimensions d'analyse.....	7
3. Modélisation du Data Warehouse.....	8
3.1. Modèle conceptuel décisionnel (schéma en étoile).....	8
3.2. Modèle logique (tables de faits et dimensions).....	9
3.2.1. Tables de faits.....	9
3.2.2. Tables de dimensions.....	10
3.3. Modèle physique dans PostgreSQL.....	11
3.4. Choix techniques.....	12
4. Mise en place de la base et chargement initial.....	13
4.1. Création de la base et des schémas.....	13
4.2. Import des données sources.....	14
4.2.1. Données de ventes - fichier Superstore.....	14
4.2.2. Données de stock - base gestionstock.....	14
4.3. Configuration des métadonnées Talend.....	15
4.3.1 Connexions aux bases PostgreSQL.....	15
4.3.2 Fichier source Superstore (File delimited).....	15
4.4. Génération de la dimension Temps (dw.dim_date).....	16
4.5. Contraintes d'intégrité et clés étrangères.....	16
5. Chaîne ETL avec Talend Open Studio.....	18
5.1. Outils et environnement technique.....	18
5.2. Jobs de staging (schéma stg).....	18
5.2.1. Job de staging des ventes : job_stg_superstore_load.....	18
5.2.2. Jobs de staging de la base gestionstock.....	19
5.3. Jobs de chargement des dimensions (dw.dim_*).....	19
5.4. Jobs de chargement des faits (dw.fact_vente et dw.fact_stock).....	20
5.4.1. Job de chargement des ventes : job_fact_vente_load.....	21
5.4.2. Job de chargement du stock : job_fact_stock_load.....	21
5.5. Job parent d'orchestration de la chaîne ETL.....	22
5.6. Gestion des erreurs, logs et rejouabilité.....	22

6. Exploitation décisionnelle du Data Warehouse.....	23
6.1. Vues OLAP ROLAP dans PostgreSQL.....	23
6.2. Analyse OLAP des ventes.....	29
6.2.1. Drill-down et roll-up sur l'axe temps.....	29
6.2.2. Slice et dice sur les produits et les clients.....	30
6.2.3. Mesures dérivées sur les ventes.....	30
6.3. Analyse OLAP du stock.....	30
6.4. Alertes décisionnelles et supervision.....	31
6.5. Exemples de requêtes OLAP.....	32
6.5.1. Drill-down / roll-up sur l'axe temps – produit.....	32
6.5.2. Slice et dice sur produits, clients et régions.....	33
6.5.3. Mesures dérivées sur les ventes.....	36
6.5.4. Requêtes OLAP sur le stock.....	39
6.5.5. Exemples de requêtes d'alerte.....	42
7. Conclusion.....	44
Annexe : Captures d'écran du processus ETL et des analyses OLAP.....	45

1. Introduction

1.1 Contexte du projet

Dans un contexte où les entreprises collectent de plus en plus de données (ventes, clients, stocks, logistique...), la capacité à centraliser, nettoyer et analyser ces informations devient un enjeu stratégique.

Le module *Data Warehouse* s'inscrit dans cette logique : il vise à mettre en place une petite plate-forme décisionnelle complète, depuis les sources opérationnelles jusqu'aux cubes d'analyse.

Dans ce projet, nous travaillons à partir de deux sources principales :

- Le fichier **Superstore** qui contient l'historique des ventes (commandes, clients, produits, livraisons, montants, remises, profits,...)
- La base **gestionstock** qui décrit les entrepôts, les fournisseurs et les niveaux de stock par produit.

L'objectif est de transformer ces données brutes en un **entrepôt de données** structuré, capable de répondre à des questions métier comme :

- Quels sont les produits les plus rentables par période et par région ?
- Comment évoluent les ventes dans le temps selon les segments de clients ?
- Quels entrepôts présentent des risques de rupture de stock ?

1.2 Objectifs du projet

Ce travail poursuit à la fois des objectifs pédagogiques et techniques :

- concevoir un **schéma en étoile** adapté aux besoins d'analyse (tables de faits et dimensions)
- mettre en place un **processus ETL** complet avec Talend Open Studio : extraction, transformation, chargement
- alimenter un **Data Warehouse** dans PostgreSQL, en séparant les zones *staging* (stg) et *décisionnelle* (dw)
- créer des **vues OLAP (ROLAP)** pour faciliter l'analyse multidimensionnelle des ventes et des stocks
- illustrer les **opérations OLAP** classiques : *drill-down*, *roll-up*, *slice*, *dice* et calcul de mesures dérivées (marge, taux de croissance, alertes sur le stock, etc.)

1.3 Périmètre fonctionnel

Le périmètre du projet couvre les axes d'analyse suivants :

- **Ventes** : montants de ventes (*Sales*), quantités, profits, remises
- **Temps** : année, mois, nom du mois, jour...
- **Produit** : catégorie, sous-catégorie, produit
- **Client** : client, segment et zone géographique (via les dimensions régionales)
- **Logistique et stock** : entrepôts, fournisseurs, niveaux de stock, dates de réapprovisionnement.

Les données sont intégrées dans une seule base décisionnelle **dw_superstore** et organisées en deux tables de faits principales :

- dw.fact_vente pour les ventes,
- dw.fact_stock pour les mouvements / niveaux de stock.

1.4 Architecture décisionnelle mise en place

Pour atteindre ces objectifs, nous avons adopté une architecture simple mais complète :

1. **Sources opérationnelles**
 - Fichier CSV *Superstore*
 - Base relationnelle gestionstock (tables entrepôts, fournisseurs, stock)
2. **Zone de staging (schéma stg)**
 - Chargement brut des données sources via des jobs Talend
 - Harmonisation minimale des types et des formats (dates, clés, textes)
3. **Zone Data Warehouse (schéma dw)**
 - Dimensions : dim_date, dim_produit, dim_client, dim_entrepot, dim_fournisseur, dim_ship_mode,...
 - Faits : fact_vente et fact_stock avec clés de substitution et mesures agrégables.
4. **Couches d'analyse OLAP (vues ROLAP)**
 - Vues d'agrégation sur les ventes par temps, produit, client, région, mode d'expédition
 - Vues d'agrégation sur le stock par temps, entrepôt, fournisseur
 - Vues d'alertes sur les ruptures potentielles de stock ou les marges faibles

2. Analyse des besoins décisionnels

2.1. Problématique métier

Les données exploitées dans ce projet proviennent d'un contexte de distribution et de vente de produits de bureau, de mobilier, de fournitures et de matériel technologique.

L'entreprise fictive gère à la fois :

- des **ventes** réalisées auprès de différents segments de clients
- des **stocks** répartis sur plusieurs entrepôts et alimentés par des fournisseurs variés

La problématique métier peut se résumer ainsi :

“Comment exploiter les données de ventes et de stocks pour mieux piloter l’activité commerciale et logistique, tout en anticipant les risques (rupture de stock, faible marge, mauvais choix de produits ou de segments) ?”

Cette question globale se décline en plusieurs sous-questions décisionnelles :

- Quels produits génèrent le plus de chiffre d'affaires et de profit, par période ?
- Quels segments de clients et quelles régions sont les plus rentables ?
- Quels entrepôts présentent des stocks insuffisants sur certains produits stratégiques ?
- Comment évoluent les ventes d'une année à l'autre (taux de croissance) ?
- Quels couples *produit × client* ou *produit × région* posent problème en termes de marge ou de stock disponible ?

Le Data Warehouse doit permettre de répondre à ces questions de manière rapide, fiable et flexible.

2.2. Questions décisionnelles principales

À partir de la problématique précédente, on peut formuler un ensemble de questions décisionnelles auquel l'entrepôt de données devra répondre, par exemple :

- **Analyse des ventes**
 - Quel est le **chiffre d'affaires** et le **profit** par année, trimestre, mois ?
 - Quelles **catégories** et **sous-catégories** de produits sont les plus performantes ?
 - Quels sont les **clients** ou **segments** les plus rentables ?

- Quelles **régions** ou **pays** présentent les meilleures performances de ventes ?
- Quel est l'impact des **remises (discount)** sur le profit ?
- **Analyse logistique et stock**
 - Quel est le **niveau de stock** par produit, entrepôt et fournisseur ?
 - Quels produits sont proches de la **rupture de stock** (stock inférieur à un seuil) ?
 - Quels **fournisseurs** alimentent les produits les plus critiques en termes de stock ?
- **Analyse temporelle et tendance**
 - Comment évoluent les ventes d'une année à l'autre (taux de croissance) ?
 - Existe-t-il des **saisonnalités** (pics et creux) par mois ou par trimestre ?

Ces besoins orientent directement la définition des mesures (indicateurs) et des dimensions dans le modèle en étoile.

2.3. Mesures identifiées

À partir des tables de faits fact_vente et fact_stock, les principaux indicateurs retenus sont :

- **Mesures liées aux ventes** (table dw.fact_vente) :
 - *Sales* : montant des ventes
 - *Quantity* : quantité vendue
 - *Profit* : profit associé à une ligne de commande ;
 - *Discount* : remise appliquée
 - *Marge %* = Profit / Sales (calculée dans les vues OLAP)
 - *Nombre de commandes* (count distinct des order_id)
- **Mesures liées aux stocks** (table dw.fact_stock) :
 - *Quantité en stock* : quantite_stock par produit, entrepôt, fournisseur et date
 - *Stock disponible par catégorie / entrepôt*
 - *Stock / Ventes* : rapport entre le stock disponible et les quantités vendues (indicateur de risque de rupture)

Certains indicateurs sont stockés directement dans les tables de faits (Sales, Quantity, Profit, Stock), tandis que d'autres sont calculés dynamiquement dans les vues ROLAP : marge %, pourcentage du total, cumul des ventes, taux de croissance, etc.

2.4. Dimensions d'analyse

Pour analyser ces indicateurs sous différents angles, plusieurs dimensions ont été définies dans le schéma dw :

- **Dimension Temps** (dw.dim_date)
 - Attributs : annee, trimestre, mois, nom_mois, jour, etc.
 - Sert de base aux hiérarchies temporelles (Année → Trimestre → Mois → Jour) et aux analyses de tendance.
- **Dimension Produit** (dw.dim_produit)
 - Attributs : category, sub_category, product_id, product_name, ...
 - Permet de comparer les performances par catégorie, sous-catégorie et produit.
- **Dimension Client** (dw.dim_client)
 - Attributs typiques : identifiant client, nom, segment, ville, État, pays, région...
 - Utilisée pour analyser les ventes par segment (Consumer, Corporate, Home Office...), par région géographique et par type de clientèle.
- **Dimension Entrepôt / Lieu de stockage** (dw.dim_entrepot)
 - Attributs : nom_entrepot, region, etc.
 - Permet d'analyser le stock disponible et les risques de rupture par entrepôt ou région logistique.
- **Dimension Fournisseur** (dw.dim_fournisseur)
 - Attributs : nom_fournisseur, pays, informations de contact...
 - Sert à suivre les fournisseurs responsables des produits en stock et à identifier les partenaires critiques.
- **Dimension Mode d'expédition** (dw.dim_ship_mode)
 - Attributs : ship_mode_name (mode de livraison : Standard Class, Second Class, etc.).
 - Utile pour analyser l'impact des choix logistiques sur les ventes et les délais de livraison.

En combinant ces dimensions avec les mesures des tables de faits, il devient possible de construire des **cubes ROLAP** permettant une analyse multidimensionnelle : par exemple, ventes par *Temps × Produit × Région*, stock par *Temps × Entrepôt × Fournisseur*, ou encore marges par *Segment de client × Catégorie de produit*.

3. Modélisation du Data Warehouse

3.1. Modèle conceptuel décisionnel (schéma en étoile)

À partir des besoins décisionnels identifiés, nous avons opté pour une **modélisation en étoile** (star schema). Ce type de modèle est particulièrement adapté aux systèmes décisionnels, car il :

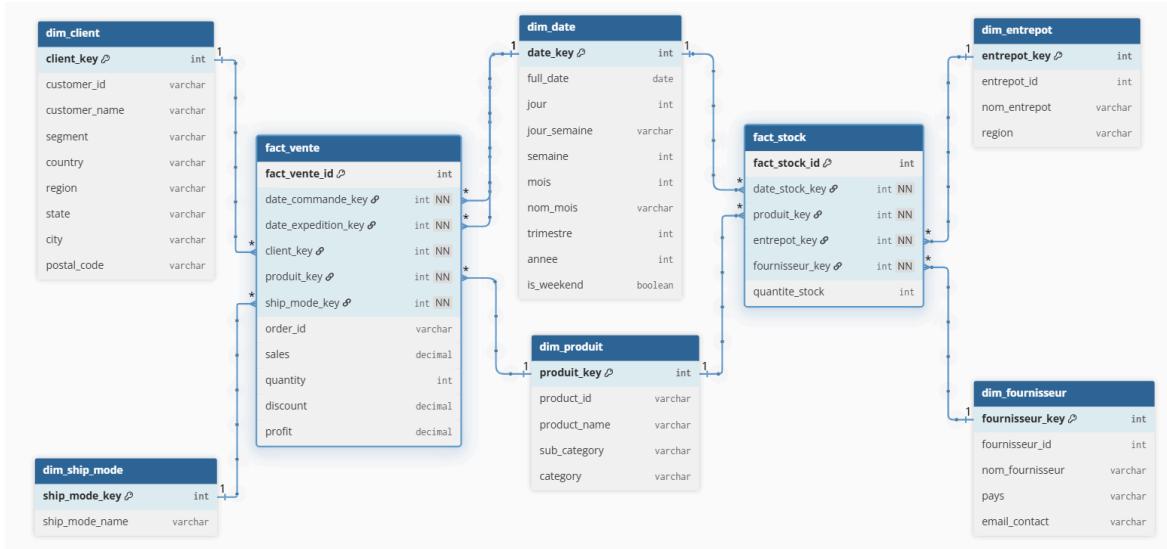
- sépare clairement les **tables de faits** (mesures chiffrées) des **tables de dimensions** (axes d'analyse)
- simplifie les requêtes d'agrégation (GROUP BY sur les dimensions)
- facilite la construction de cubes OLAP (ROLAP dans notre cas).

Le cœur du schéma est constitué de deux tables de faits :

- **dw.fact_vente** : enregistre les ventes détaillées, au niveau de la **ligne de commande** ;
- **dw.fact_stock** : enregistre les informations de **stock disponible** par produit, entrepôt, fournisseur et date.

Autour de ces faits, plusieurs dimensions décrivent le contexte :

- **dw.dim_date** : dimension Temps commune aux deux faits
- **dw.dim_produit** : description des produits (catégorie, sous-catégorie, nom, identifiant)
- **dw.dim_client** : description des clients et de leur segment
- **dw.dim_entrepot** : description des lieux de stockage
- **dw.dim_fournisseur** : description des fournisseurs
- **dw.dim_ship_mode** : description des modes d'expédition / livraison.



Graphiquement, on obtient un schéma en étoile où les deux faits sont au centre, reliés par des **clés de substitution (surrogate keys)** aux différentes dimensions.

3.2. Modèle logique (tables de faits et dimensions)

Le **modèle logique** précise la structure des tables (colonnes, clés, relations), sans entrer encore dans les détails techniques propres à PostgreSQL.

3.2.1. Tables de faits

- **Table dw.fact_vente**
 - **Grain (niveau de granularité)** : une ligne par **détail de commande** (Order Line).
 - Principales clés :
 - fact_vente_id (clé primaire technique)
 - date_commande_key → FK vers dw.dim_date
 - client_key → FK vers dw.dim_client
 - produit_key → FK vers dw.dim_produit
 - ship_mode_key → FK vers dw.dim_ship_mode
 - Principales mesures :
 - sales (montant de vente)
 - quantity (quantité vendue)
 - discount (remise)
 - profit (profit)
 - ainsi que des identifiants de suivi (ex : order_id).

- **Table dw.fact_stock**
 - **Grain** : une ligne par (**date, produit, entrepôt, fournisseur**), ce qui permet de suivre l'évolution du stock disponible dans le temps.
 - Principales clés :
 - fact_stock_id (clé primaire technique)
 - date_key → FK vers dw.dim_date
 - produit_key → FK vers dw.dim_produit
 - entrepot_key → FK vers dw.dim_entrepot
 - fournisseur_key → FK vers dw.dim_fournisseur
 - Principale mesure :
 - quantite_stock (stock disponible à la date considérée).

Cet agencement permet de croiser les **ventes** et le **stock** à travers des dimensions communes (Temps, Produit, Région / Entrepôt, Fournisseur).

3.2.2. Tables de dimensions

Les dimensions sont relativement dénormalisées pour faciliter la lecture et la performance des requêtes OLAP.

- **dw.dim_date**
 - Rôle : dimension Temps.
 - Colonnes typiques :
 - date_key (clé technique, PK)
 - full_date (date au format DATE)
 - annee, trimestre, mois, nom_mois, jour, éventuellement semaine...
 - Utilisée par fact_vente et fact_stock pour toutes les analyses temporelles.
- **dw.dim_produit**
 - Rôle : décrire les produits vendus / stockés.
 - Colonnes :
 - produit_key (PK)
 - product_id (identifiant opérationnel, issu de Superstore / gestionstock)
 - product_name
 - category, sub_category
 - autres attributs possibles selon les besoins.

- **dw.dim_client**
 - Rôle : décrire les clients et leurs segments.
 - Colonnes possibles :
 - client_key (PK)
 - customer_id (id source)
 - customer_name
 - segment
 - city, state, country, region...
- **dw.dim_entrepot**
 - Rôle : représenter les lieux de stockage.
 - Colonnes :
 - entrepot_key (PK)
 - entrepot_id (id source)
 - nom_entrepot
 - region (région logistique).
- **dw.dim_fournisseur**
 - Rôle : représenter les fournisseurs.
 - Colonnes :
 - fournisseur_key (PK)
 - fournisseur_id (id source)
 - nom_fournisseur
 - pays
 - email_contact...
- **dw.dim_ship_mode**
 - Rôle : décrire le mode de livraison (logistique).
 - Colonnes :
 - ship_mode_key (PK)
 - ship_mode_name (Standard Class, Second Class, etc.).

Ce modèle logique en étoile garantit une bonne lisibilité et une compatibilité naturelle avec les vues OLAP (ROLAP) implémentées par la suite.

3.3. Modèle physique dans PostgreSQL

Le **modèle physique** correspond à la traduction effective du modèle logique dans PostgreSQL.

Deux schémas principaux sont utilisés :

- **Schéma stg (Staging)**
 - Contient les tables de staging, proches des sources :
 - stg_superstore (chargée depuis le CSV Superstore),
 - stg_entrepots, stg_fournisseurs, stg_stock (chargées depuis la base gestionstock).
 - Les données y sont chargées par Talend avec un minimum de transformations (types, formats de dates...).
- **Schéma dw (Data Warehouse)**
 - Contient les tables de dimensions et de faits :
 - dw.dim_date, dw.dim_produit, dw.dim_client, dw.dim_entrepot, dw.dim_fournisseur, dw.dim_ship_mode,
 - dw.fact_vente, dw.fact_stock.
 - Des **clés primaires** et **clés étrangères** sont définies pour assurer l'intégrité référentielle entre faits et dimensions.

L'ensemble est regroupé dans la même base PostgreSQL, ce qui facilite l'administration, les sauvegardes et les connexions depuis les outils d'analyse.

3.4. Choix techniques

Plusieurs décisions techniques ont été prises pour simplifier et fiabiliser l'implémentation :

- **SGBD** : choix de **PostgreSQL** comme moteur de base de données :
 - open source, robuste et compatible avec les fonctionnalités analytiques (agrégations, fonctions fenêtre, vues, etc.)
 - facilement accessible depuis Talend Open Studio
- **Séparation stg / dw** :
 - le schéma stg sert de zone tampon entre les sources et l'entrepôt
 - le schéma dw ne contient que des données propres, structurées selon le modèle en étoile
- **Clés de substitution (surrogate keys)** dans les dimensions :
 - utilisation de colonnes entières auto-incrémentées (*_key) pour les jointures, indépendamment des identifiants opérationnels (product_id, customer_id, etc.)

- **Types de données :**
 - types numériques adéquats pour les montants et quantités (NUMERIC, INTEGER) ;
 - types DATE pour les dates, réutilisés à travers la dimension dim_date ;
 - VARCHAR pour les libellés (noms de produits, clients, entrepôts...).
- **Vues OLAP (ROLAP) :**
 - plutôt que de créer des cubes OLAP dans un serveur dédié, le choix a été de construire des **vues d'agrégation** dans PostgreSQL (CREATE VIEW).
 - ces vues jouent le rôle de **cubes ROLAP**, directement interrogables en SQL ou depuis un outil de visualisation.

Ce modèle physique sert de base aux étapes suivantes du rapport, consacrées à la chaîne ETL implémentée dans Talend, puis à la construction des vues OLAP et aux opérations analytiques réalisées sur ces cubes.

4. Mise en place de la base et chargement initial

4.1. Crédation de la base et des schémas

La première étape technique a consisté à créer l'environnement décisionnel dans PostgreSQL.

1. Crédation de la base de données décisionnelle

Une base dédiée a été créée pour le Data Warehouse, par exemple :

- Base : **dw_superstore**
- 2. Cette base regroupe à la fois la zone de staging et le schéma décisionnel final.
- 3. **Crédation des schémas logiques**

À l'intérieur de cette base, deux schémas principaux ont été définis :

- **Schéma stg (staging) :**
 - utilisé comme zone intermédiaire pour charger les données brutes depuis les sources (CSV, base gestionstock) ;
 - permet de séparer clairement les données sources des données décisionnelles.
- **Schéma dw (data warehouse) :**
 - contient les tables de dimensions (dim_*) et de faits (fact_*) selon le modèle en étoile décrit précédemment ;
 - c'est le schéma cible sur lequel s'appuient les vues OLAP et les requêtes d'analyse.

Ces schémas ont été créés via des scripts SQL exécutés dans pgAdmin, afin d'avoir une structure de base propre avant de lancer la chaîne ETL dans Talend.

4.2. Import des données sources

Deux grandes familles de données sont exploitées : les données de ventes (Superstore) et les données de stock (gestionstock).

4.2.1. Données de ventes - fichier Superstore

Les données de ventes sont fournies sous forme d'un fichier **CSV** (Sample – Superstore), qui contient les informations suivantes :

- commandes (Order ID, Order Date, Ship Date, Ship Mode) ;
- clients (Customer ID, Customer Name, Segment, Pays, Région, Ville, etc.) ;
- produits (Product ID, Product Name, Category, Sub-Category) ;
- mesures associées (Sales, Quantity, Discount, Profit, ...).

Le fichier CSV est d'abord chargé dans une table de staging, par exemple :

- stg.stg_superstore

Ce chargement est réalisé automatiquement via un job Talend (tFileInputDelimited → tPostgresqlOutput) plutôt que manuellement, afin d'industrialiser le processus et de pouvoir le rejouer facilement.

4.2.2. Données de stock - base gestionstock

Les données de stock proviennent d'une base opérationnelle **gestionstock**, fournie sous forme d'un script SQL.

Cette base contient principalement :

- la table **entrepots** (entrepot_id, nom_entrepot, region)
- la table **fournisseurs** (fournisseur_id, nom_fournisseur, pays, email_contact)
- la table **stock** (product_id, nom_produit, fournisseur_id, entrepot_id, quantite_stock, date_dernier_reapprovisionnement)

Le script a été **adapté et exécuté dans PostgreSQL**, puis les données ont été transférées vers des tables de staging spécifiques, par exemple :

- stg.stg_entrepots

- stg.stg_fournisseurs
- stg.stg_stock

Là aussi, les chargements sont automatisés par des jobs Talend, afin de pouvoir rafraîchir les données sans intervention manuelle.

4.3. Configuration des métadonnées Talend

4.3.1 Connexions aux bases PostgreSQL

Dans Talend Open Studio, nous avons d'abord défini les connexions aux bases PostgreSQL dans le volet **Metadata → Db Connections** :

- **Postgres_dw_superstore0.1** : connexion vers le data warehouse dw_superstore (hôte localhost, port 5432, base dw_superstore, utilisateur postgres). Cette connexion est utilisée par les jobs de chargement des dimensions et des faits, ainsi que par les vues OLAP.
- **Postgres_gestionstock_0.1** : connexion vers la base opérationnelle gestionstock qui contient les tables sources entrepots, fournisseurs et stock. Cette connexion est utilisée dans les jobs de staging pour extraire les données sources.

Toutes les composantes Talend (tPostgresqlInput, tPostgresqlOutput, tDBRow, etc.) s'appuient sur ces connexions définies en **Repository** afin de centraliser les paramètres techniques (URL, login, mot de passe) et de faciliter la réutilisation dans plusieurs jobs.

4.3.2 Fichier source Superstore (File delimited)

Sous **Metadata → File delimited**, nous avons défini le schéma du fichier CSV Superstore_csv 0.1 correspondant au fichier de ventes “Superstore”. Le format du fichier (séparateur ; ou , , encodage, présence de l'en-tête, types de colonnes) est paramétré une seule fois dans la métadonnée.

Ensuite, cette définition de fichier est utilisée en mode **Repository** dans le composant tFileInputDelimited du job job_stg_superstore_load. Cela permet de garantir la cohérence du schéma entre le fichier source et la table de staging stg.stg_superstore.

4.4. Génération de la dimension Temps (`dw.dim_date`)

La dimension Temps est un élément central de l'entrepôt, car elle est référencée à la fois par les ventes (`fact_vente`) et par le stock (`fact_stock`).

Plutôt que d'importer un fichier externe, la table `dw.dim_date` a été générée directement dans PostgreSQL à l'aide d'un script SQL basé sur la fonction `generate_series`.

Les principales étapes sont :

1. Définir l'intervalle temporel

- Début : une date antérieure à la première commande / premier réapprovisionnement (par exemple 2014-01-01) ;
- Fin : une date ultérieure à la dernière date de vente ou de réapprovisionnement (par exemple 2025-12-31), pour couvrir une plage suffisamment large.

2. Générer une ligne par jour

- utilisation de `generate_series(date_debut, date_fin, interval '1 day')` pour créer une date par jour ;

3. Calculer les attributs dérivés pour chaque date :

- annee, trimestre, mois, nom_mois, jour, éventuellement semaine, jour de la semaine, etc.
- ces attributs sont stockés directement dans la table `dw.dim_date` pour simplifier les requêtes d'analyse (pas besoin de recalculer dans chaque SELECT).

Cette même dimension est ensuite référencée :

- par `dw.fact_vente` via deux clés de dates : `date_commande_key` (date de commande) et `date_expedition_key` (date d'expédition) ;
- par `dw.fact_stock` via `date_key` (date d'état du stock / réapprovisionnement).

4.5. Contraintes d'intégrité et clés étrangères

Une fois les tables créées et les premiers chargements effectués, des **contraintes d'intégrité** ont été définies afin de garantir la cohérence des données dans le Data Warehouse :

1. Clés primaires

- chaque dimension possède une clé primaire de type entier (surrogate key)
 - date_key pour dw.dim_date
 - produit_key pour dw.dim_produit
 - client_key pour dw.dim_client
 - entrepot_key pour dw.dim_entrepot
 - fournisseur_key pour dw.dim_fournisseur
 - ship_mode_key pour dw.dim_ship_mode
- Les tables de faits possèdent également une clé primaire technique (fact_vente_id, fact_stock_id) pour l'unicité des lignes.

2. Clés étrangères

- des contraintes FOREIGN KEY lient les faits aux dimensions correspondantes, par exemple :
 - fact_vente.date_commande_key → dim_date.date_key
 - fact_vente.client_key → dim_client.client_key
 - fact_vente.produit_key → dim_produit.produit_key
 - fact_vente.ship_mode_key → dim_ship_mode.ship_mode_key
 - fact_stock.date_key → dim_date.date_key
 - fact_stock.produit_key → dim_produit.produit_key
 - fact_stock.entrepot_key → dim_entrepot.entrepot_key
 - fact_stock.fournisseur_key → dim_fournisseur.fournisseur_key

3. Contrôles de cohérence

- l'ordre d'exécution des jobs ETL respecte cette intégrité :
 - chargement d'abord des dimensions (DIM), puis des faits (FACT) ;
- les erreurs éventuelles (valeurs orphelines, types incompatibles) sont gérées au niveau de Talend (filtres, rejets) ou détectées lors de l'insertion dans PostgreSQL.

Grâce à ces contraintes, le Data Warehouse fournit une base fiable sur laquelle on peut construire les **vues OLAP (ROLAP)** et réaliser des analyses multidimensionnelles sans risque majeur d'incohérence entre faits et dimensions.

5. Chaîne ETL avec Talend Open Studio

5.1. Outils et environnement technique

Pour l'implémentation de la chaîne ETL, nous avons utilisé :

- **Talend Open Studio for Data Integration**
 - Version : TOS_DI-20211109_1610-V8.0.1
 - Outil principal pour l'**extraction**, la **transformation** et le **chargement** des données (ETL).
- **PostgreSQL 17.5**
 - SGBD cible pour le Data Warehouse, hébergeant les schémas stg et dw dans la base dw_superstore.
- **Connecteurs Talend**
 - tFileInputDelimited pour lire le fichier CSV Superstore ;
 - tPostgresqlConnection et tPostgresqlOutput pour se connecter à PostgreSQL et insérer les données ;
 - tPrejob pour l'initialisation des traitements dans les jobs.

Les connexions à PostgreSQL (base dw_superstore et éventuellement gestionstock adaptée) ont été définies dans la **rubrique Metadata** de Talend, afin de pouvoir être réutilisées dans plusieurs jobs sans reconfigurer les paramètres techniques.

5.2. Jobs de staging (schéma stg)

La première couche de l'ETL consiste à charger les données brutes dans le schéma stg. Chaque source a son propre job de staging.

5.2.1. Job de staging des ventes : job_stg_superstore_load

Ce job permet de charger le fichier CSV Superstore dans la table stg.stg_superstore.

- **Composants principaux :**
 - tFileInputDelimited :
 - lit le fichier CSV (Sample – Superstore)
 - configure le séparateur, l'en-tête, le type de chaque colonne (dates, nombres, texte, ...)

- tPostgresqlOutput :
 - insère les lignes dans la table stg.stg_superstore du schéma stg de dw_superstore.

Ce job constitue le point d'entrée des données de ventes dans la chaîne ETL.

5.2.2. Jobs de staging de la base gestionstock

À partir du script SQL de la base gestionstock, les tables suivantes ont été reconstruites dans PostgreSQL :

- entrepots
- fournisseurs
- stock

Ensuite, trois jobs Talend assurent le passage de ces tables vers le schéma stg :

- **job_stg_entrepots_load**
 - Source : table entrepots
 - Cible : stg.stg_entrepots
- **job_stg_fournisseurs_load**
 - Source : table fournisseurs
 - Cible : stg.stg_fournisseurs
- **job_stg_stock_load**
 - Source : table stock
 - Cible : stg.stg_stock

Chaque job suit la même structure générale :

- tPostgresqlInput pour lire les données source
- tPostgresqlOutput pour insérer dans les tables de staging.

Cette couche de staging permet de séparer clairement les données brutes des traitements de transformation ultérieurs.

5.3. Jobs de chargement des dimensions (dw.dim_*)

Une fois les données disponibles dans le schéma stg, des jobs spécifiques sont responsables de la construction et de l'alimentation des dimensions dans le schéma dw.

En pratique, la logique est la même pour chaque dimension :

- lire les données dans stg.* ou dans les tables sources ;
- éliminer les doublons ;
- générer une clé de substitution (*_key) si nécessaire ;
- charger dans dw.dim_*.

Par exemple :

- **Dimension Produit : dw.dim_produit**
 - Source : colonnes produit (Product ID, Product Name, Category, Sub-Category) issues de stg.stg_superstore ;
 - Traitement :
 - déduplication par Product ID ;
 - construction de la ligne de dimension (clé technique + attributs descriptifs) ;
 - Cible : dw.dim_produit.
- **Dimension Client – dw.dim_client**
 - Source : informations client dans stg.stg_superstore ;
 - Traitement :
 - déduplication par Customer ID ;
 - récupération du segment, pays, région, ville, etc. ;
 - Cible : dw.dim_client.
- **Dimension Entrepôt / Fournisseur / Ship Mode**
 - Source : stg.stg_entrepots, stg.stg_fournisseurs, et les champs de livraison du fichier Superstore ;
 - Cible : dw.dim_entrepot, dw.dim_fournisseur, dw.dim_ship_mode.

La dimension Temps dw.dim_date est générée directement dans PostgreSQL via script SQL, mais elle pourrait également être intégrée à un job Talend dédié si l'on souhaitait automatiser totalement sa création.

5.4. Jobs de chargement des faits (dw.fact_vente et dw.fact_stock)

Après la disponibilité des dimensions, les tables de faits peuvent être alimentées.

5.4.1. Job de chargement des ventes : job_fact_vente_load

Ce job construit la table dw.fact_vente à partir des données détaillées des commandes.

- **Source principale** : stg.stg_superstore
- **Étapes principales** :
 1. Lecture des lignes de commandes (Order ID, Order Date, Customer, Product, Sales, Quantity, Discount, Profit, Ship Mode...).
 2. Jointures (via tMap) avec les dimensions déjà chargées :
 - dw.dim_date pour obtenir date_commande_key à partir d'Order Date ;
 - dw.dim_client pour client_key ;
 - dw.dim_produit pour produit_key ;
 - dw.dim_ship_mode pour ship_mode_key.
 3. Construction de la ligne de fait :
 - clés étrangères vers les dimensions ;
 - mesures : sales, quantity, discount, profit ;
 - identifiants de suivi (par exemple order_id).
 4. Insertion dans dw.fact_vente via tPostgresqlOutput.

Le grain de la table de faits est ainsi fixé à la **ligne de commande**.

5.4.2. Job de chargement du stock : job_fact_stock_load

Ce job construit la table dw.fact_stock à partir des informations de stock issues de stg.stg_stock.

- **Source principale** : stg.stg_stock
- **Étapes principales** :
 1. Lecture des lignes de stock (product_id, entrepot_id, fournisseur_id, quantite_stock, date_dernier_reapprovisionnement).
 2. Jointures (via tMap) avec :
 - dw.dim_produit (correspondance par product_id → produit_key)
 - dw.dim_entrepot (entrepot_id → entrepot_key)

- dw.dim_fournisseur (fournisseur_id → fournisseur_key);
 - dw.dim_date (date_dernier_reapprovisionnement → date_key).
3. Construction de la ligne de fait :
 - date_key, produit_key, entrepot_key, fournisseur_key
 - mesure quantite_stock.
 4. Insertion dans dw.fact_stock.

Ce job permet de lier les informations de stock aux mêmes dimensions que les ventes, ce qui rend possible des analyses croisées (ventes vs stock).

5.5. Job parent d'orchestration de la chaîne ETL

Pour automatiser l'exécution de toute la chaîne ETL dans le bon ordre, un **job parent** a été conçu dans Talend (par exemple job_parent_dw_superstore).

Ce job a pour rôle d'enchaîner :

1. Les jobs de staging (job_stg_*);
2. Les jobs de dimensions (job_dim_*);
3. Les jobs de faits (job_fact_vente_load, job_fact_stock_load).

La logique générale est la suivante :

- Un composant **tPrejob** initialise la séquence (connexion, paramètres globaux...).
- À partir de ce tPrejob, des sous-jobs sont déclenchés dans l'ordre souhaité (via les liens OnSubjobOk / OnComponentOk) :
 1. chargement du staging (Superstore + gestionstock) ;
 2. mise à jour des dimensions ;
 3. chargement des faits fact_vente puis fact_stock.

Cette approche garantit qu'à chaque exécution du job parent, l'entrepôt est rafraîchi de manière cohérente, avec les dimensions toujours prêtes avant les faits.

5.6. Gestion des erreurs, logs et rejouabilité

Même si le projet reste de taille pédagogique, plusieurs bonnes pratiques ont été adoptées :

- **Séparation claire STG / DW**
 - En cas de problème, il est possible de vider et recharger uniquement les tables de staging, sans casser le modèle décisionnel.
- **Rejouabilité des jobs**
 - les jobs Talend sont conçus pour pouvoir être relancés (mode INSERT/TRUNCATE selon les cas) ;
 - l'ordre d'exécution est fixé par le job parent pour éviter les inconsistances (par exemple des faits sans dimensions associées).
- **Validation dans PostgreSQL**
 - des requêtes de contrôle (COUNT, MIN/MAX dates, comparaison avec les sources) sont exécutées pour vérifier le volume et la cohérence des données ;
 - des contraintes de clés étrangères assurent un dernier niveau de validation (impossibilité d'insérer des clés inexistantes).

Dans une version industrielle, cette partie pourrait être complétée par des composants Talend supplémentaires (`tLogCatcher`, `tFileOutputDelimited` pour les erreurs, scheduling automatique, etc.). Ici, l'objectif principal était de démontrer la maîtrise de la chaîne ETL de bout en bout.

6. Exploitation décisionnelle du Data Warehouse

Cette dernière partie illustre comment le data warehouse `dw_superstore` peut être utilisé pour répondre à des besoins décisionnels concrets.

Les vues OLAP créées (ventes par temps et produit, ventes par mode de livraison, stock par temps et entrepôt, etc.) servent de base à différentes opérations analytiques de type **ROLAP** : drill-down, roll-up, slice, dice et calcul de mesures dérivées.

6.1. Vues OLAP ROLAP dans PostgreSQL

1. Vue `dw.v_ventes_temps_produit`

- **Rôle** : analyse des ventes dans le temps par famille de produits.
- **Dimensions** : Temps (année, mois, nom_mois), Produit (category, sub_category).
- **Mesures** : total_sales, total_quantity, total_profit, avg_discount.
- **Exemple d'usage** : suivre l'évolution mensuelle du chiffre d'affaires par catégorie de produit.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

Servers(1) PostgreSQL 17 Databases(3) dw_superstore Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postgres Login/Group Roles Tablespaces

```

1 v CREATE OR REPLACE VIEW dw.v_ventes_temps_produit AS
2   SELECT
3     d.annee,
4     d.mois,
5     d.nom_mois,
6     p.category,
7     p.sub_category,
8     SUM(f.sales) AS total_sales,
9     SUM(f.quantity) AS total_quantity,
10    SUM(f.profit) AS total_profit,
11    AVG(f.discount) AS avg_discount
12   FROM dw.fact_vente f
13  JOIN dw.dim_date d ON d.date_key = f.date_commande_key
14  JOIN dw.dim_produit p ON p.produit_key = f.produit_key

```

Data Output Messages Notifications

CREATE VIEW

Query returned successfully in 85 msec.

Total rows: Query complete 00:00:00.085 CRLF Ln 26, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

Servers(1) PostgreSQL 17 Databases(3) dw_superstore Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postgres Login/Group Roles Tablespaces

```

1 SELECT * FROM dw.v_ventes_temps_produit LIMIT 10;
2

```

Data Output Messages Notifications

	annee	mois	nom_mois	category	sub_category	total_sales	total_quantity	total_profit	avg_discount
1	2014	1	January	Furniture	Bookcases	1832.34	78	-2563.08	0.5666666666666666
2	2014	1	January	Furniture	Chairs	51392.16	246	12771.92	0.0000000000000000
3	2014	1	January	Furniture	Furnishings	9350.02	324	771.86	0.1192307692307692
4	2014	1	January	Furniture	Tables	2664.00	24	-133.20	0.2000000000000000
5	2014	1	January	Office Supplies	Appliances	778.32	48	77.88	0.2000000000000000
6	2014	1	January	Office Supplies	Art	1786.62	278	526.02	0.0157894736842105
7	2014	1	January	Office Supplies	Binders	16840.08	312	5887.54	0.3047619047619047
8	2014	1	January	Office Supplies	Envelopes	1230.20	110	602.66	0.0000000000000000
9	2014	1	January	Office Supplies	Labels	466.20	66	163.74	0.1200000000000000
10	2014	1	January	Office Supplies	Paper	1331.64	234	535.84	0.0689655172410555

Activate Windows
Go to Settings to activate Windows.

Total rows: 10 Query complete 00:00:00.215 CRLF Ln 2, Col 1

2. Vue dw.v_ventes_client_region

- **Rôle** : analyser les ventes par région et type de client.
- **Dimensions** : Temps (année, mois, nom_mois), Client (region, segment).
- **Mesures** : total_sales, total_quantity, total_profit.

- Exemple d'usage : comparer la performance des segments de clients entre régions.

pgAdmin 4

File Object Tools Edit View Window Help

Servers(1) PostgreSQL 17 Databases(3) dw_superstore

dw Superstore

Casts Catalogs Event Triggers Extensions Foreign Data W Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postres Login/Group Roles Tablespaces

Query Query History

```

1 CREATE OR REPLACE VIEW dw.v_ventes_client_region AS
2   SELECT
3     d.annee,
4     d.mois,
5     d.nom_mois,
6     c.region,
7     c.segment,
8     SUM(f.sales) AS total_sales,
9     SUM(f.quantity) AS total_quantity,
10    SUM(f.profit) AS total_profit
11   FROM dw.fact_vente f
12  JOIN dw.dim_date d ON d.date_key = f.date_commande_key
13  JOIN dw.dim_client c ON c.client_key = f.client_key
14  GROUP BY
15    d.annee,
16    d.mois
  
```

Data Output Messages Notifications

CREATE VIEW

Query returned successfully in 95 msec.

Total rows: Query complete 00:00:00.095

Activate Windows

✓ Query returned successfully in 95 msec. X

CRLF Ln 25, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Servers(1) PostgreSQL 17 Databases(3) dw_superstore

dw Superstore

Casts Catalogs Event Triggers Extensions Foreign Data W Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postres Login/Group Roles Tablespaces

Query Query History

```

1 SELECT * FROM dw.v_ventes_client_region LIMIT 10;
2
  
```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

	annee	mois	nom_mois	region	segment	total_sales	total_quantity	total_profit
	integer	integer	character varying(50)	character varying(50)	character varying(50)	numeric	bigint	numeric
1	2014	1	January	Central	Consumer	15051.98	368	1569.68
2	2014	1	January	Central	Corporate	3139.64	68	373.54
3	2014	1	January	Central	Home Office	22525.58	212	6681.26
4	2014	1	January	East	Consumer	18338.12	424	3162.22
5	2014	1	January	East	Corporate	3721.12	90	180.32
6	2014	1	January	East	Home Office	2575.24	104	-35.56
7	2014	1	January	South	Consumer	16019.78	296	2667.64
8	2014	1	January	South	Corporate	5912.12	112	1107.28
9	2014	1	January	South	Home Office	30433.94	168	9820.16
10	2014	1	January	West	Consumer	16279.46	390	1136.98

Activate Windows

Go to Settings to activate Windows.

Total rows: 10 Query complete 00:00:00.223

CRLF Ln 2, Col 1

3. Vue dw.v_stock_temps_entrepot

- **Rôle** : suivi du stock par entrepôt dans le temps.
- **Dimensions** : Temps (année, mois, nom_mois), Entrepôt (region, nom_entrepot).
- **Mesures** : total_stock.
- **Exemple d'usage** : détecter les entrepôts avec sur-stock ou sous-stock sur une période.

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a tree structure of databases, schemas, and tables. In the central query editor, a SQL script is being run to create a view. The message area at the bottom indicates the query was successful and completed in 78 msec. A tooltip provides additional context about the message.

```
CREATE OR REPLACE VIEW dw.v_stock_temps_entrepot AS
SELECT
    d.annee,
    d.mois,
    d.nom_mois,
    e.region,
    e.nom_entrepot,
    SUM(fs.quantite_stock) AS total_stock
FROM dw.fact_stock fs
JOIN dw.dim_date d ON d.date_key = fs.date_key
JOIN dw.dim_entrepot e ON e.entrepot_key = fs.entrepot_key
GROUP BY
    d.annee,
    d.mois,
    d.nom_mois,
    e.region,
    e.nom_entrepot
```

Activate Windows
✓ Query returned successfully in 78 msec. X

The screenshot shows the pgAdmin 4 interface. On the left is the object browser with a tree view of databases, schemas, and tables. The main area is a query editor window titled 'dw_superstore/postgres@PostgreSQL 17'. It contains a SQL query and its results. The query is:

```
1 SELECT * FROM dw.v_stock_temps_entrepot LIMIT 10;
```

The results table has the following columns: annee, mois, nom_mois, region, nom_entrepot, and total_stock. The data is as follows:

	annee	mois	nom_mois	region	nom_entrepot	total_stock
1	2024	1	January	Centre	Entrepot Centre	85
2	2024	1	January	Est	Entrepot Est	335
3	2024	1	January	Nord	Entrepot Nord	370
4	2024	1	January	Ouest	Entrepot Ouest	35
5	2024	1	January	Sud	Entrepot Sud	1128
6	2024	2	February	Centre	Entrepot Centre	74
7	2024	2	February	Est	Entrepot Est	940
8	2024	2	February	Global	Entrepot Global	269
9	2024	2	February	Nord	Entrepot Nord	649
10	2024	2	February	Ouest	Entrepot Ouest	263

4. Vue dw.v_stock_temps_fournisseur_produit

- **Rôle :** analyse du stock par fournisseur et produit.
- **Dimensions :** Temps (année, mois, nom_mois), Fournisseur (pays, nom_fournisseur), Produit (category, sub_category).
- **Mesures :** total_stock.
- **Exemple d'usage :** identifier les fournisseurs critiques pour certaines familles de produits.

The screenshot shows the pgAdmin 4 interface. The object browser on the left shows the database structure. The main query editor window contains the SQL code for creating a view:

```
1 CREATE OR REPLACE VIEW dw.v_stock_temps_fournisseur_produit AS
2 SELECT
3     d.annee,
4     d.mois,
5     d.nom_mois,
6     f.pays,
7     f.nom_fournisseur,
8     p.category,
9     p.sub_category,
10    SUM(fs.quantite_stock) AS total_stock
11   FROM dw.fact_stock      fs
12  JOIN dw.dim_date        d ON d.date_key       = fs.date_key
13  JOIN dw.dim_fournisseur f ON f.fournisseur_key = fs.fournisseur_key
14  JOIN dw.dim_produit      p ON p.produit_key    = fs.produit_key
15 GROUP BY
16     d.annee,
17     d.mois,
18     d.nom_mois,
```

The results pane at the bottom shows the message: "CREATE VIEW" and "Query returned successfully in 100 msec."

Object Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

File Object Tools Edit View Window Help

Servers(1) PostgreSQL 17 Databases(3) dw_superstore Casts Catalogs Event Triggers Extensions Foreign Data W Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postgres Login/Group Roles Tablespaces

Query Query History

```
1 SELECT * FROM dw.v_stock_temps_fournisseur_produit LIMIT 10;
```

Data Output Messages Notifications

	annee	mois	nom_mois	pays	nom_fournisseur	category	sub_category	total_stock
	integer	integer	character varying(20)	character varying(100)	character varying(150)	character varying(100)	character varying(100)	bigint
1	2024	1	January	Allemagne	Papeterie Monde	Furniture	Bookcases	4
2	2024	1	January	Canada	Bureautique Inc.	Furniture	Tables	1
3	2024	1	January	Chine	Fournitures Asie	Office Supplies	Binders	33
4	2024	1	January	Espagne	Bureautique Europe	Technology	Phones	3
5	2024	1	January	États-Unis	Fournitures USA	Office Supplies	Art	37
6	2024	1	January	France	Mobilier Europe	Office Supplies	Envelopes	18
7	2024	1	January	France	Mobilier Europe	Office Supplies	Labels	25
8	2024	1	January	France	Mobilier Europe	Office Supplies	Paper	68
9	2024	1	January	Japon	Techno Supplies	Office Supplies	Appliances	4
10	2024	2	February	Afrique du Sud	Mobilier Afrique	Furniture	Chairs	4

5. Vue dw.v_ventes_temps_livraison

- **Rôle :** analyser l'impact du mode de livraison sur les ventes.
- **Dimensions :** Temps (année, mois, nom_mois), Livraison (ship_mode_name).
- **Mesures :** total_sales, total_quantity, total_profit, avg_discount.
- **Exemple d'usage :** comparer la performance commerciale des différents modes de livraison.

Object Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

File Object Tools Edit View Window Help

Servers(1) PostgreSQL 17 Databases(3) dw_superstore Casts Catalogs Event Triggers Extensions Foreign Data W Languages Publications Schemas(3) dw public stg Subscriptions gestionstock postgres Login/Group Roles Tablespaces

Query Query History

```
1 CREATE OR REPLACE VIEW dw.v_ventes_temps_livraison AS
2 SELECT
3     d.annee,
4     d.mois,
5     d.nom_mois,
6     sm.ship_mode_name      AS ship_mode,
7     SUM(f.sales)           AS total_sales,
8     SUM(f.quantity)        AS total_quantity,
9     SUM(f.profit)          AS total_profit,
10    AVG(f.discount)       AS avg_discount
11   FROM dw.dim_date      f
12   JOIN dw.dim_date      d ON d.date_key      = f.date_commande_key
13   JOIN dw.dim_ship_mode sm ON sm.ship_mode_key = f.ship_mode_key
14 GROUP BY
15     d.annee,
16     d.mois,
17     d.nom_mois,
18     sm.ship_mode_name;
```

Data Output Messages Notifications

CREATE VIEW

Query returned successfully in 66 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: Query complete 00:00:00.066 CRLF Ln 19, Col 1

The screenshot shows the pgAdmin 4 interface. On the left, the object browser displays a tree structure of databases, schemas, and tables. The main area shows a query window with the following SQL code:

```

1 < SELECT *
2   FROM dw.v_ventes_temps_livraison
3   LIMIT 10;

```

The results are displayed in a table with the following columns and data:

	annee	mois	nom_mois	ship_mode	total_sales	total_quantity	total_profit	avg_discount
1	2014	1	January	First Class	3160.10	142	-1975.00	0.26842105263157894737
2	2014	1	January	Second Class	35271.06	704	6917.52	0.13033707865168539326
3	2014	1	January	Standard Class	117391.08	1676	27580.74	0.09596774193548387097
4	2014	2	February	First Class	8821.60	260	3433.32	0.03529411764705882353
5	2014	2	February	Second Class	5492.34	342	1600.56	0.04516129032258064516
6	2014	2	February	Standard Class	34134.34	976	5711.02	0.18151260504201680672
7	2014	3	March	First Class	26908.20	504	6555.44	0.19870129870129870130
8	2014	3	March	Same Day	8120.64	210	363.54	0.28484848484848484848
9	2014	3	March	Second Class	41690.96	996	8335.26	0.118881118881118881112
10	2014	3	March	Standard Class	531947.74	3736	-679.00	0.17333333333333333333

At the bottom, it says "Total rows: 10 Query complete 00:00:00.143".

6.2. Analyse OLAP des ventes

Les vues basées sur la table de faits dw.fact_vente (par exemple dw.v_ventes_temps_produit et dw.v_ventes_temps_livraison) permettent d'explorer les ventes selon plusieurs axes :

- **Temps** : année, mois, jour
- **Produit** : catégorie, sous-catégorie
- **Client** : client_key (extensible vers une dimension client)
- **Mode de livraison** : ship_mode

6.2.1. Drill-down et roll-up sur l'axe temps

Le modèle temporel et la vue dw.v_ventes_temps_produit permettent :

- le **roll-up** : passer d'un niveau détaillé à une vision synthétique, par exemple des ventes agrégées *par mois* ou *par année* ;
- le **drill-down** : descendre au détail, par exemple des ventes *du mois* vers les ventes *par jour* ou *par sous-catégorie*.

Concrètement, un décideur peut commencer par une vision globale des ventes annuelles par catégorie, puis zoomer sur une année spécifique, ensuite sur un mois, puis sur les produits les plus contributeurs.

6.2.2. Slice et dice sur les produits et les clients

Les dimensions **Produit** et **Client** permettent d'appliquer :

- un **slice** : filtrer le cube sur une valeur d'une dimension, par exemple « ventes de la catégorie *Furniture* uniquement » ou « ventes du segment *Corporate* » (si une dimension client est ajoutée)
- un **dice** : combiner plusieurs filtres simultanés, par exemple :
 - période limitée (année 2016)
 - catégorie = *Office Supplies*
 - mode de livraison = *Second Class*

Ce type d'analyse permet d'identifier rapidement les combinaisons *produit* × *période* × *mode de livraison* les plus rentables ou les plus risquées.

6.2.3. Mesures dérivées sur les ventes

À partir des mesures de base (sales, quantity, profit, discount), plusieurs indicateurs dérivés peuvent être calculés :

- **Marge en valeur** : profit
- **Taux de marge (%)** : profit / sales
- **Réduction moyenne (%)** : moyenne de discount sur la période
- **Part de marché relative** d'une catégorie : ventes d'une catégorie / ventes totales
- **Taux de croissance des ventes** d'une période à l'autre (mois N vs mois N-1, année N vs N-1)

Ces indicateurs peuvent être calculés dans les vues SQL, dans un outil de reporting ou dans des requêtes analytiques spécifiques.

6.3. Analyse OLAP du stock

La table de faits dw.fact_stock et la vue dw.v_stock_temps_entrepot permettent de suivre l'évolution du stock :

- par **produit** (clé produit_key) ;

- par **fournisseur** (clé fournisseur_key) ;
- par **entrepôt** (clé entrepot_key) ;
- dans le **temps** (clé date_key → dw.dim_date).

Quelques scénarios typiques :

- Suivi du **stock disponible** par entrepôt et par catégorie de produits.
- Identification des **produits à risque de rupture** (quantité faible sur plusieurs entrepôts).
- Analyse du **stock moyen** sur une période donnée (par mois, par trimestre).
- Comparaison **stock / ventes** pour détecter les produits sur-stockés ou sous-stockés.

Des mesures dérivées peuvent être définies, par exemple :

- **Stock moyen mensuel** d'un produit.
- **Ratio stock / ventes** : quantite_stock / total_sales (en reliant fact_stock et fact_vente par produit et période).

6.4. Alertes décisionnelles et supervision

À partir des vues analytiques, il est possible de créer des vues ou requêtes SQL dédiées aux **alertes métier**, par exemple :

- **Rupture ou quasi-rupture de stock** : produits dont la quantité est inférieure à un seuil (par exemple 10 unités) sur un ou plusieurs entrepôts.
- **Produits à faible marge** : lignes de vente où profit / sales est inférieur à un seuil (par exemple 5 %).
- **Clients ou segments non rentables** : si une dimension client est ajoutée, repérer les clients avec un volume de ventes important mais une marge très faible.

Ces vues peuvent être interrogées régulièrement par un outil externe (script, Talend job, outil de reporting) pour générer :

- des **rapports quotidiens/hebdomadaires**,
- des **notifications** (emails, tableaux de bord, etc.),
- ou alimenter un futur **système d'alertes en temps quasi-réel**.

6.5. Exemples de requêtes OLAP

Cette section illustre concrètement l'utilisation des vues analytiques du schéma dw à travers quelques requêtes SQL typiques (drill-down, roll-up, slice, dice, mesures dérivées, alertes, etc.).

Toutes les requêtes suivantes peuvent être exécutées directement dans pgAdmin.

6.5.1. Drill-down / roll-up sur l'axe temps – produit

Roll-up : ventes annuelles par catégorie

```
-- Ventes annuelles par catégorie
SELECT
    annee,
    category,
    SUM(total_sales)      AS sales_annee_cat,
    SUM(total_profit)     AS profit_annee_cat
FROM dw.v_ventes_temps_produit
GROUP BY annee, category
ORDER BY annee, category;
```

The screenshot shows the pgAdmin 4 interface. On the left is the object browser with a tree view of servers, databases, and tables. The central area is a query editor window titled 'dw_superstore/postgres@PostgreSQL 17*'. It contains the SQL code for a roll-up query. Below the code is a table preview showing the results of the query. The table has columns: annee (integer), category (character varying(100)), sales_annee_cat (numeric), and profit_annee_cat (numeric). The data shows annual sales and profits for Furniture, Office Supplies, and Technology categories across years 2014, 2015, 2016, and 2017. At the bottom of the table, it says 'Total rows: 12' and 'Query complete 00:00:00.754'. A message at the bottom right says 'Activate Windows Go to Settings to activate Windows.' The status bar at the bottom right shows 'CRLF Ln 10, Col 1'.

	annee	category	sales_annee_cat	profit_annee_cat
1	2014	Furniture	1534012.78	50198.52
2	2014	Office Supplies	1564599.98	229500.86
3	2014	Technology	2265143.70	314969.08
4	2015	Furniture	1498470.88	62994.62
5	2015	Office Supplies	1318339.92	239811.48
6	2015	Technology	2262246.70	473008.34
7	2016	Furniture	1964161.84	47612.56
8	2016	Office Supplies	2023859.54	417192.64
9	2016	Technology	2916120.84	474042.40
10	2017	Furniture	1864988.02	13070.84
11	2017	Office Supplies	2804951.26	419414.92
12	2017	Technology	2804951.26	419414.92

Cette requête agrège les ventes au niveau **année × catégorie**. Elle illustre l'opération de **roll-up**.

Drill-down : détail mensuel pour une année et une catégorie

```
-- Détail mensuel pour l'année 2016 et la catégorie Furniture
SELECT
    annee,
    mois,
    nom_mois,
    category,
    sub_category,
    SUM(total_sales) AS sales_mois_souscat,
    SUM(total_profit) AS profit_mois_souscat
FROM dw.v_ventes_temps_produit
WHERE annee = 2016
    AND category = 'Furniture'
GROUP BY annee, mois, nom_mois, category, sub_category
ORDER BY mois, sub_category;
```

The screenshot shows the pgAdmin 4 interface. On the left is the object browser with a tree view of servers, databases, and tables. The central area contains a query editor window titled 'dw_superstore/postgres@PostgreSQL 17*'. The query is the one shown above. Below the query is a results grid displaying the monthly sales and profits for the Furniture category in 2016, broken down by month and sub-category. The results show positive sales and negative profit for some categories like Tables.

	annee	mois	nom_mois	category	sub_category	sales_mois_souscat	profit_mois_souscat
1	2016	1	January	Furniture	Bookcases	582.96	93.24
2	2016	1	January	Furniture	Chairs	26229.12	546.42
3	2016	1	January	Furniture	Furnishings	8000.72	1331.90
4	2016	1	January	Furniture	Tables	11221.56	-5174.80
5	2016	2	February	Furniture	Chairs	13697.48	2282.04
6	2016	2	February	Furniture	Furnishings	6177.96	1156.44
7	2016	2	February	Furniture	Tables	9966.14	-416.04
8	2016	3	March	Furniture	Bookcases	6446.12	-3247.76
9	2016	3	March	Furniture	Chairs	45148.26	-1574.68
10	2016	3	March	Furniture	Furnishings	31308.80	5310.48
11	2016	3	March	Furniture	Tables	2890.54	-4199.22
						11600.00	1110.00

Ici, on descend du niveau **année × catégorie** vers **mois × sous-catégorie** : c'est un **drill-down** sur le temps et le produit.

6.5.2. Slice et dice sur produits, clients et régions

Slice : ventes de la catégorie “Office Supplies” uniquement

On applique un **slice** en filtrant une seule valeur de la dimension Produit.

```
-- Slice sur la catégorie Office Supplies
SELECT
    annee,
    category,
    SUM(total_sales) AS sales_cat,
    SUM(total_profit) AS profit_cat
FROM dw.v_ventes_temps_produit
WHERE category = 'Office Supplies'
GROUP BY annee, category
ORDER BY annee;
```

The screenshot shows the pgAdmin 4 interface. On the left is the Explorer pane with a tree view of servers, databases, and tables. The main area contains a query editor window with the previously provided SQL code. Below the code is a Data Output grid showing the results:

	annee	category	sales_cat	profit_cat
1	2014	Office Supplies	1564599.98	229500.86
2	2015	Office Supplies	1318339.92	239811.48
3	2016	Office Supplies	2023859.54	417192.64
4	2017	Office Supplies	2804951.26	419414.92

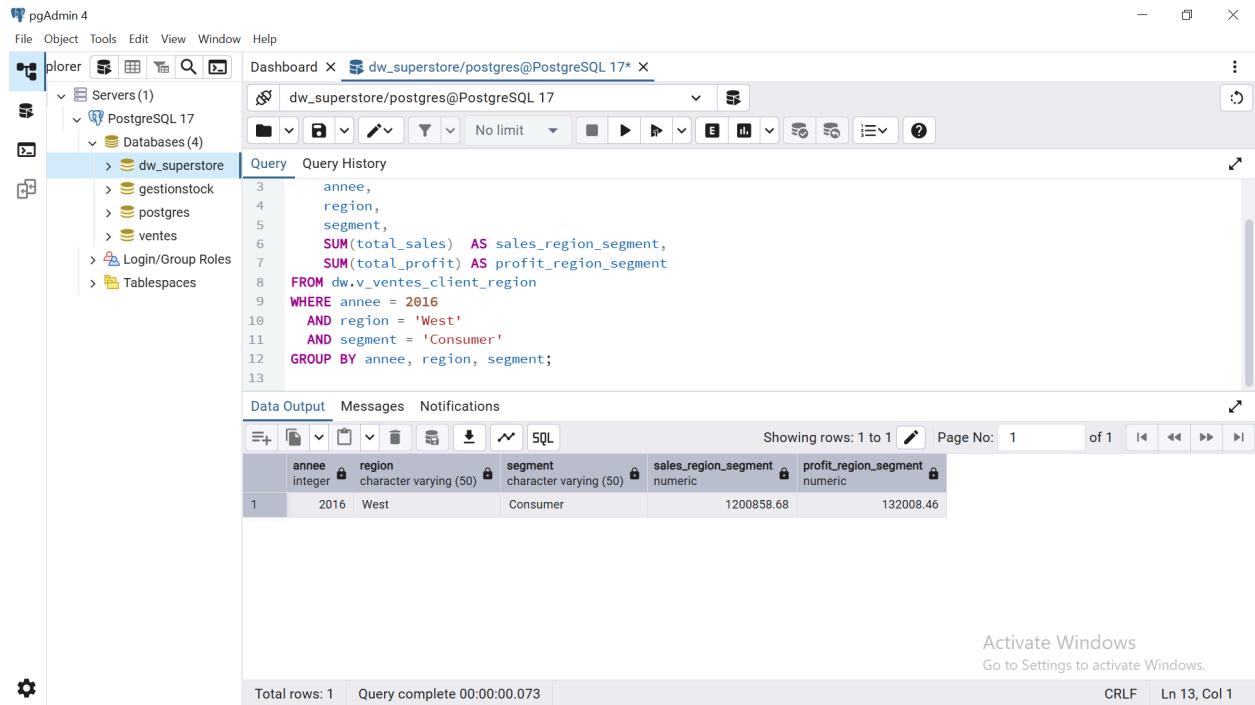
At the bottom of the pgAdmin window, there are status messages: "Activate Windows Go to Settings to activate Windows.", "Total rows: 4 Query complete 00:00:00.100", "CRLF Ln 10, Col 16", and a gear icon.

Dice : combinaison année + région + segment

```
-- Dice : année 2016, région West, segment Consumer
SELECT
    annee,
    region,
    segment,
    SUM(total_sales) AS sales_region_segment,
    SUM(total_profit) AS profit_region_segment
FROM dw.v_ventes_client_region
WHERE annee = 2016
    AND region = 'West'
    AND segment = 'Consumer'
GROUP BY annee, region, segment;
```

Cette requête combine plusieurs filtres simultanés (année, région, segment) :

c'est une opération de **dice** dans le cube Ventes.



```
pgAdmin 4
File Object Tools Edit View Window Help
Explorer Dashboard dw_superstore/postgres@PostgreSQL 17*
Servers(1)
  PostgreSQL 17
    Databases(4)
      > dv_superstore
      > gestionstock
      > postgres
      > ventes
    > Login/Group Roles
    > Tablespaces
Query Query History
3   annee,
4   region,
5   segment,
6   SUM(total_sales) AS sales_region_segment,
7   SUM(total_profit) AS profit_region_segment
8   FROM dw.v_ventes_client_region
9   WHERE annee = 2016
10  AND region = 'West'
11  AND segment = 'Consumer'
12 GROUP BY annee, region, segment;
13

Data Output Messages Notifications
Showing rows: 1 to 1 | Page No: 1 of 1 | << << >> >> | ↻
annee region segment sales_region_segment profit_region_segment
1 2016 West Consumer 1200858.68 132008.46

Activate Windows
Go to Settings to activate Windows.

Total rows: 1 Query complete 00:00:00.073 CRLF Ln 13, Col 1
```

Slice sur le mode de livraison

```
-- Ventes par année et mode de livraison pour "Second Class"
SELECT
  annee,
  ship_mode,
  SUM(total_sales) AS sales_mode,
  SUM(total_profit) AS profit_mode
FROM dw.v_ventes_temps_livraison
WHERE ship_mode = 'Second Class'
GROUP BY annee, ship_mode
ORDER BY annee;
```

On analyse l'impact d'un **mode de livraison** spécifique sur les ventes.

```

-- Ventes par année et mode de livraison pour "Second Class"
SELECT
    annee,
    ship_mode,
    SUM(total_sales) AS sales_mode,
    SUM(total_profit) AS profit_mode
FROM dw.v_ventes_temps_livraison
WHERE ship_mode = 'Second Class'
GROUP BY annee, ship_mode
ORDER BY annee;

```

The screenshot shows the pgAdmin 4 interface with the following details:

- Servers:** dw_superstore (selected)
- Databases:** dw_superstore (selected)
- Query History:** Contains the executed SQL query.
- Data Output:** Shows the results of the query in a table format.
- Table Headers:** annee, ship_mode, sales_mode, profit_mode
- Table Data:**

annee	ship_mode	sales_mode	profit_mode
2014	Second Class	1044846.16	90307.06
2015	Second Class	929345.14	111874.86
2016	Second Class	1508274.78	220867.32
2017	Second Class	1619995.76	223597.10

- Messages:** Successfully run. Total query runtime: 102 ms; 4 rows affected.
- Log:** CRLF Ln 11, Col 1

6.5.3. Mesures dérivées sur les ventes

Taux de marge (%) par année et catégorie

```

-- Taux de marge par année et catégorie
SELECT
    annee,
    category,
    SUM(total_sales) AS sales_cat,
    SUM(total_profit) AS profit_cat,
    ROUND(
        SUM(total_profit) / NULLIF(SUM(total_sales), 0) * 100,
        2
    ) AS taux_marge_pct
FROM dw.v_ventes_temps_produit
GROUP BY annee, category
ORDER BY annee, category;

```

On calcule ici un **taux de marge** en pourcentage pour chaque combinaison année × catégorie.

```

-- Taux de marge par année et catégorie
SELECT
    annee,
    category,
    SUM(total_sales) AS sales_cat,
    SUM(total_profit) AS profit_cat,
    ROUND(
        SUM(total_profit) / NULLIF(SUM(total_sales), 0) * 100,
        2
    ) AS taux_marge_pct
FROM dw.v_ventes_temps_produit

```

	annee	category	sales_cat	profit_cat	taux_marge_pct
1	2014	Furniture	1534012.78	50198.52	3.27
2	2014	Office Supplies	1564599.98	229500.86	14.67
3	2014	Technology	2265143.70	314969.08	13.91
4	2015	Furniture	1498470.88	62994.62	4.20
5	2015	Office Supplies	1318339.92	239811.48	18.19
6	2015	Technology	2262246.70	473008.34	20.91
7	2016	Furniture	1964161.84	47612.56	2.42
8	2016	Office Supplies	2023859.54	417192.64	20.61
	Total rows: 12				
					Query complete 00:00:00.141

Part de chaque catégorie dans le total annuel

```

-- Part de catégorie dans le chiffre d'affaires annuel
WITH ventes_cat AS (
    SELECT
        annee,
        category,
        SUM(total_sales) AS sales_cat
    FROM dw.v_ventes_temps_produit
    GROUP BY annee, category
)
SELECT
    annee,
    category,
    sales_cat,
    ROUND(
        sales_cat
        / SUM(sales_cat) OVER (PARTITION BY annee) * 100,
        2
    ) AS part_cat_annee_pct
FROM ventes_cat
ORDER BY annee, category;

```

On utilise ici une **fonction fenêtre** pour calculer la **part de chaque catégorie** dans le total annuel.

pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(4)

dw_superstore gestionstock postgres ventes Login/Group Roles Tablespaces

Query Query History

```

1 -- Part de catégorie dans le chiffre d'affaires annuel
2 WITH ventes_cat AS (
3     SELECT
4         annee,
5         category,
6         SUM(total_sales) AS sales_cat
7     FROM dw.v_ventes_temps_produit
8     GROUP BY annee, category
9 )
10 SELECT
11     annee,

```

Data Output Messages Notifications

Showing rows: 1 to 12 Page No: 1 of 1

	annee	category	sales_cat	part_cat_annee_pct
1	2014	Furniture	1534012.78	28.60
2	2014	Office Supplies	1564599.98	29.17
3	2014	Technology	2265143.70	42.23
4	2015	Furniture	1498470.88	29.50
5	2015	Office Supplies	1318339.92	25.96
6	2015	Technology	2262246.70	44.54
7	2016	Furniture	1964161.84	28.45
8	2016	Office Supplies	2023859.54	29.31
	Total rows: 12			
				Query complete 00:00:00.111

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 20, Col 26

Taux de croissance des ventes par catégorie d'une année à l'autre

```

-- Taux de croissance des ventes par catégorie
WITH ventes_cat AS (
    SELECT
        annee,
        category,
        SUM(total_sales) AS sales_cat
    FROM dw.v_ventes_temps_produit
    GROUP BY annee, category
)
SELECT
    annee,
    category,
    sales_cat,
    LAG(sales_cat) OVER (PARTITION BY category ORDER BY annee) AS sales_annee_prec,
    ROUND(
        (sales_cat
        - LAG(sales_cat) OVER (PARTITION BY category ORDER BY annee))
        / NULLIF(LAG(sales_cat) OVER (PARTITION BY category ORDER BY annee), 0)
        * 100,
        2
    ) AS taux_croissance_pct
FROM ventes_cat
ORDER BY category, annee;

```

Cette requête calcule le **taux de croissance des ventes** d'une année à l'autre pour chaque catégorie.

```
-- Taux de croissance des ventes par catégorie
WITH ventes_cat AS (
    SELECT
        annee,
        category,
        SUM(total_sales) AS sales_cat
    FROM dw.v_ventes_temps_produit
    GROUP BY annee, category
)
SELECT
    annee,
    category,
    sales_cat,
    sales_annee_prec,
    taux_croissance_pct
FROM (
    SELECT
        annee,
        category,
        sales_cat,
        LAG(sales_cat) OVER (PARTITION BY category ORDER BY annee) AS sales_annee_prec,
        ((sales_cat - sales_annee_prec) / sales_annee_prec) * 100 AS taux_croissance_pct
    FROM ventes_cat
) AS subquery
ORDER BY annee, category;
```

	annee	category	sales_cat	sales_annee_prec	taux_croissance_pct
1	2014	Furniture	1534012.78	[null]	[null]
2	2015	Furniture	1498470.88	1534012.78	-2.32
3	2016	Furniture	1964161.84	1498470.88	31.08
4	2017	Furniture	1864988.02	1964161.84	-5.05
5	2014	Office Supplies	1564599.98	[null]	[null]
6	2015	Office Supplies	1318339.92	1564599.98	-15.74
7	2016	Office Supplies	2023859.54	1318339.92	53.52
8	2017	Office Supplies	2804951.26	2023859.54	38.59
	Total rows: 12		Query complete 00:00:00.114		

6.5.4. Requêtes OLAP sur le stock

Stock annuel par région et entrepôt

```
-- Stock total par année, région et entrepôt
SELECT
    annee,
    region,
    nom_entrepot,
    SUM(total_stock) AS stock_annee_entrepot
FROM dw.v_stock_temps_entrepot
GROUP BY annee, region, nom_entrepot
ORDER BY annee, region, nom_entrepot;
```

Cette requête permet de suivre l'évolution du **stock agrégé** par entrepôt et par région.

pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(4)

dw_superstore gestionstock postgres ventes Login/Group Roles Tablespaces

Query Query History

```

1 -- Stock total par année, région et entrepôt
2 SELECT
3     annee,
4     region,
5     nom_entrepot,
6     SUM(total_stock) AS stock_annee_entrepot
7 FROM dw.v_stock_temps_entrepot
8 GROUP BY annee, region, nom_entrepot
9 ORDER BY annee, region, nom_entrepot;

```

Data Output Messages Notifications

	annee	region	nom_entrepot	stock_annee_entrepot
1	2024	Centre	Entrepot Centre	358
2	2024	Est	Entrepot Est	2957
3	2024	Global	Entrepot Global	1124
4	2024	Nord	Entrepot Nord	6578
5	2024	Ouest	Entrepot Ouest	1179
6	2024	Sud	Entrepot Sud	6565

Showing rows: 1 to 6 Page No: 1 of 1

Total rows: 6 Query complete 00:00:00.081 CRLF Ln 9, Col 38

Activate Windows
Go to Settings to activate Windows.

Produits en risque de rupture pour une année donnée

```

-- Produits en quasi-rupture de stock en 2024 (seuil = 20)
SELECT
    annee,
    nom_entrepot,
    region,
    total_stock
FROM dw.v_stock_temps_entrepot
WHERE annee = 2024
    AND total_stock < 20
ORDER BY total_stock ASC;

```

Elle identifie les entrepôts dont la quantité est inférieure à un **seuil critique**.

The screenshot shows the pgAdmin 4 interface. On the left is the object browser with a tree view of servers, databases, and tables. The main area is a query editor titled "dw_superstore/postgres@PostgreSQL 17". It contains a SQL query and its results. The query retrieves products near stockout in 2024. The results table has four columns: annee, nom_entrepot, region, and total_stock, with one row showing data for Entrepot Centre.

	annee	nom_entrepot	region	total_stock
1	2024	Entrepot Centre	Centre	14

Stock par fournisseur et catégorie

```
-- Analyse du stock par fournisseur et catégorie
SELECT
    annee,
    nom_fournisseur,
    category,
    sub_category,
    SUM(total_stock) AS stock_fournisseur_cat
FROM dw.v_stock_temps_fournisseur_produit
GROUP BY annee, nom_fournisseur, category, sub_category
ORDER BY annee, nom_fournisseur, category, sub_category;
```

Cette requête combine les dimensions **temps**, **fournisseur**, **produit** pour analyser l'exposition au stock.

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers(1) > PostgreSQL 17 > Databases(4) > dw_superstore', the 'Query' tab is selected. The main area displays a SQL query:

```

-- Analyse du stock par fournisseur et catégorie
SELECT
    annee,
    nom_fournisseur,
    category,
    sub_category,
    SUM(total_stock) AS stock_fournisseur_cat
FROM dw.v_stock_temps_fournisseur_produit
GROUP BY annee, nom_fournisseur, category, sub_category
ORDER BY annee, nom_fournisseur, category, sub_category;

```

Below the query, the 'Data Output' tab is active, showing a table with 17 rows of data. The columns are: annee, nom_fournisseur, category, sub_category, and stock_fournisseur_cat. The data includes entries like '2024 Bureautique Europe Technology Accessories 1041' and '2024 Mobilier Afrique Furniture Chairs 125'. A message at the bottom right says 'Activate Windows Go to Settings to activate Windows.'

6.5.5. Exemples de requêtes d'alerte

Alerte “faible marge” sur les ventes

```

-- Alerte : sous-catégories à faible marge (< 5 %) en 2016
WITH ventes_cat AS (
    SELECT
        annee,
        category,
        sub_category,
        SUM(total_sales) AS sales_souscat,
        SUM(total_profit) AS profit_souscat
    FROM dw.v_ventes_temps_produit
    WHERE annee = 2016
    GROUP BY annee, category, sub_category
)
SELECT
    annee,
    category,
    sub_category,
    sales_souscat,
    profit_souscat,
    ROUND(
        profit_souscat / NULLIF(sales_souscat, 0) * 100,
        2
    ) AS taux_marge_pct
FROM ventes_cat

```

```

WHERE profit_souscat / NULLIF(sales_souscat, 0) < 0.05
ORDER BY taux_marge_pct;

```

Cette requête isole les **sous-catégories à très faible marge**, candidates à une action corrective (augmentation de prix, réduction des remises, etc.).

```

-- Alerte : sous-catégories à faible marge (< 5 %) en 2016
WITH ventes_cat AS (
    SELECT
        annee,
        category,
        sub_category,
        SUM(total_sales) AS sales_souscat,
        SUM(total_profit) AS profit_souscat
    FROM dw.v_ventes_temps_produit
    WHERE annee = 2016
    GROUP BY annee, category, sub_category
)
SELECT
    annee,
    category,
    sub_category
    sales_souscat
    profit_souscat
    taux_marge_pct

```

	annee	category	sub_category	sales_souscat	profit_souscat	taux_marge_pct
1	2016	Furniture	Tables	674123.78	-45019.84	-6.68
2	2016	Furniture	Bookcases	214722.56	-9870.78	-4.60
3	2016	Office Supplies	Supplies	209115.34	-952.66	-0.46
4	2016	Technology	Machines	644587.80	15612.34	2.42

Alerte “stock critique” par entrepôt

```

-- Alerte : entrepôts en stock critique (total_stock < 10)
SELECT
    annee,
    mois,
    nom_mois,
    region,
    nom_entrepot,
    total_stock
FROM dw.v_stock_temps_entrepot
WHERE total_stock < 10
ORDER BY annee, mois, region, nom_entrepot;

```

Cette requête peut être utilisée comme base d'une **vue d'alerte** interrogée régulièrement par un job planifié.

The screenshot shows the pgAdmin 4 interface. In the left sidebar, under 'Servers(1) > PostgreSQL 17 > Databases(4)', the 'dw_superstore' database is selected. The main area displays a SQL query in the 'Query' tab:

```

1 -- Alerte : entrepôts en stock critique (total_stock < 10)
2 SELECT
3     annee,
4     mois,
5     nom_mois,
6     region,
7     nom_entrepot,
8     total_stock
9 FROM dw.v_stock_temps_entrepot
10 WHERE total_stock < 10
11 ORDER BY annee, mois, region, nom_entrepot;

```

Below the query, the 'Data Output' tab is active, showing the schema of the result:

	annee	mois	nom_mois	region	nom_entrepot	total_stock
	integer	integer	character varying (20)	character varying (100)	character varying (100)	bigint

At the bottom, it says 'Total rows: 0 Query complete 00:00:00.080' and 'Activate Windows Go to Settings to activate Windows.'

7. Conclusion

Ce projet avait pour objectif de concevoir et d'implémenter un entrepôt de données décisionnel dédié au pilotage des ventes et des stocks, à partir de deux sources hétérogènes : le fichier CSV Superstore pour les ventes et la base gestionstock pour les informations de stock et de fournisseurs. Le travail réalisé a permis de mettre en place une architecture complète, depuis l'intégration des données brutes jusqu'à l'analyse OLAP.

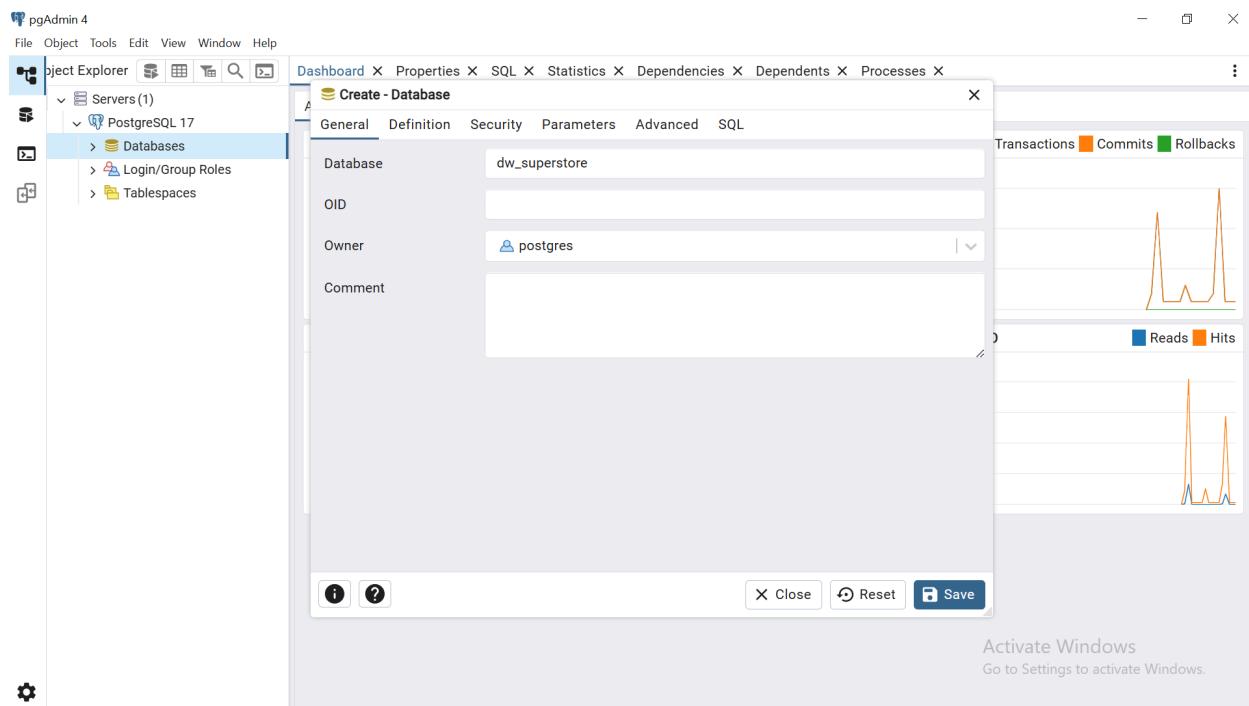
Dans une première étape, nous avons étudié les données sources et défini le grain de l'entrepôt ainsi que les tables de faits dw.fact\vente et dw.fact\stock, entourées de dimensions conformes (temps, produit, client, mode de livraison, entrepôt, fournisseur, région, etc.). Une dimension Temps générique dw.dim_date a été générée directement dans PostgreSQL à l'aide d'un script SQL, ce qui permet de mutualiser la gestion du temps pour l'ensemble des faits.

Dans une deuxième étape, nous avons construit la chaîne ETL avec Talend Open Studio : alimentation des tables de staging, chargement des dimensions, puis peuplement des tables de faits. Un job parent orchestre l'exécution de l'ensemble des sous-jobs, ce qui facilite l'automatisation future des traitements.

Enfin, plusieurs vues analytiques (ROLAP) ont été créées dans le schéma dw afin d'explorer les ventes et les stocks selon différents axes (temps, produit, client, région, entrepôt, fournisseur, mode de livraison). Ces vues servent de base à des requêtes OLAP permettant de réaliser des opérations de drill-down, roll-up, slice et dice, mais aussi de calculer des indicateurs dérivés (taux de marge, ratio stock/ventes, croissance des ventes, etc.) et de détecter des situations d'alerte (faible marge, risque de rupture de stock).

Annexe : Captures d'écran du processus ETL et des analyses OLAP

Dans cette section, nous illustrons, à l'aide de captures d'écran, toutes les étapes principales de mise en place de l'entrepôt de données (PostgreSQL, Talend).



pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(2) dw_superstore

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions postgres Login/Group Roles Tablespaces

Query History

```
1 CREATE SCHEMA stg;
2 CREATE SCHEMA dw;
```

Data Output Messages Notifications

CREATE SCHEMA

Query returned successfully in 67 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: 1 Query complete 00:00:00.067 CRLF Ln 3, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(2) dw_superstore

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions postgres Login/Group Roles Tablespaces

Query History

```
1 CREATE TABLE stg.test_connection (
2   id int PRIMARY KEY,
3   name varchar(50)
4 );
5
6 INSERT INTO stg.test_connection VALUES (1, 'hello_dw');
7
8 SELECT * FROM stg.test_connection;
```

Data Output Messages Notifications

	id	[PK] integer	name	character varying (50)
1	1		hello_dw	

Showing rows: 1 to 1 Page No: 1 of 1

Activate Windows
Go to Settings to activate Windows.

Total rows: 1 Query complete 00:00:00.424 CRLF Ln 1, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(2) dw_superstore

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions postgres Login/Group Roles Tablespaces

Query History Scratch Pad

```

SET search_path TO dw;

-- 1) Dimension Date

CREATE TABLE dim_date (
    date_key      int PRIMARY KEY,          -- format : YYYYMMDD
    full_date     date        NOT NULL,
    jour          int,
    jour_semaine varchar(20),
    semaine       int,
    mois          int,
    nom_mois     varchar(20),
    trimestre    int,
);

Data Output Messages Notifications
CREATE TABLE

Query returned successfully in 96 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: 0 Query complete 00:00:00.096 CRLF Ln 10, Col 29
```

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

Servers(1) PostgreSQL 17 Databases(2) dw_superstore

Casts Catalogs Event Triggers Extensions Foreign Data Wrappers Languages Publications Schemas Subscriptions postgres Login/Group Roles Tablespaces

Query History Scratch Pad

```

-- 8) Fact STOCK
-- grain : 1 ligne = stock d'un produit dans un entrepôt à une date

CREATE TABLE fact_stock (
    fact_stock_id   serial PRIMARY KEY,
    date_stock_key  int NOT NULL,
    produit_key     int NOT NULL,
    entrepot_key    int NOT NULL,
    fournisseur_key int NOT NULL,
    quantite_stock  int,
);

CONSTRAINT fk_factstock_date
    FOREIGN KEY (date_stock_key)
```

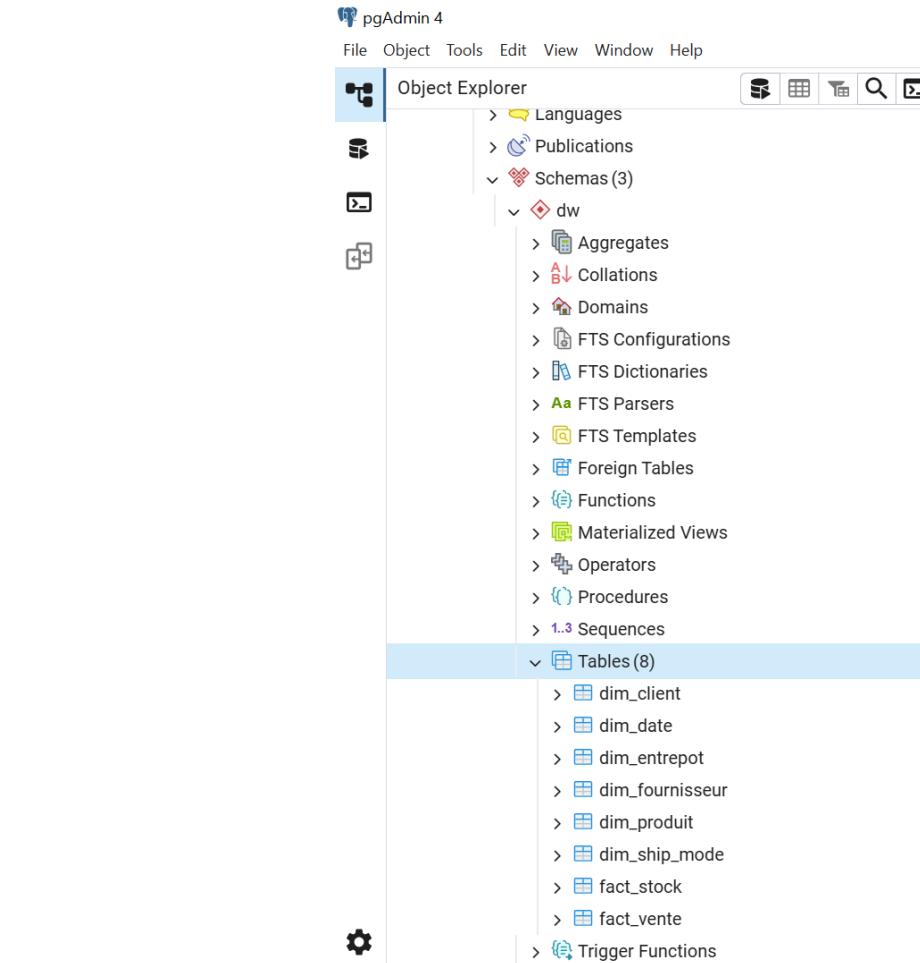
Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 96 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: 0 Query complete 00:00:00.096 CRLF Ln 8, Col 30



```

SET search_path TO stg;
CREATE TABLE stg.stg_superstore (
    row_id int,
    order_id varchar(50),
    order_date date,
    ship_date date,
    ship_mode varchar(50),
    customer_id varchar(50),
    customer_name varchar(100),
    segment varchar(50),
    country varchar(50),
    ...
);

```

The screenshot shows the pgAdmin 4 interface with the title bar "pgAdmin 4". The menu bar includes File, Object, Tools, Edit, View, Window, Help. The toolbar has icons for New, Open, Save, Print, Find, and Export. The Object Explorer sidebar lists "Servers (1)", "PostgreSQL 17", "Databases (2)", and "dw_superstore". Under "dw_superstore", the "Schemas (3)" section is expanded to show "dw". The "Tables (8)" section is also expanded, listing "dim_client", "dim_date", "dim_entrepot", "dim_fournisseur", "dim_produit", "dim_ship_mode", "fact_stock", and "fact_vente". The main area is the Query Editor, which displays the SQL code for creating the "stg.stg_superstore" table. The code is as follows:

```

1 SET search_path TO stg;
2
3 CREATE TABLE stg.stg_superstore (
4     row_id int,
5     order_id varchar(50),
6     order_date date,
7     ship_date date,
8     ship_mode varchar(50),
9     customer_id varchar(50),
10    customer_name varchar(100),
11    segment varchar(50),
12    country varchar(50),
13    ...
14 );

```

The Query Editor also shows the "Data Output" tab with the message "Query returned successfully in 64 msec." and the "Activate Windows" watermark at the bottom right.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers(1)
 - PostgreSQL 17
 - Databases(2)
 - dw_superstore

Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17* X

dw_superstore/postgres@PostgreSQL 17

Query Query History Execute script F5

```
1 v CREATE TABLE stg.stg_entrepots (
2   entrepot_id int PRIMARY KEY,
3   nom_entrepot varchar(100),
4   region varchar(100)
5 );
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 68 msec.

Activate Windows

✓ Query returned successfully in 68 msec. X

CRLF Ln 5, Col 3

Servers > PostgreSQL 17 > Databases > dw_superstore > Schemas > dw 00:00:00.068

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers(1)
 - PostgreSQL 17
 - Databases(2)
 - dw_superstore

Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17* X

dw_superstore/postgres@PostgreSQL 17

Query Query History Execute script F5

```
1 v CREATE TABLE stg.stg_fournisseurs (
2   fournisseur_id int PRIMARY KEY,
3   nom_fournisseur varchar(150),
4   pays varchar(100),
5   email_contact varchar(150)
6 );
7
```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 119 msec.

Activate Windows

✓ Query returned successfully in 119 msec. X

CRLF Ln 7, Col 1

Servers > PostgreSQL 17 > Databases > dw_superstore > Schemas > dw 00:00:00.119

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- Servers(1)
 - PostgreSQL 17
 - Databases(2)
 - dw_superstore

Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17*

dw_superstore/postgres@PostgreSQL 17

Query Query History Execute script F5

```
1 v CREATE TABLE stg.stg_stock (
2   product_id           varchar(50),
3   nom_produit          varchar(200),
4   fournisseur_id       int,
5   entrepot_id          int,
6   quantite_stock        int,
7   date_dernier_reapprovisionnement date
8 );
```

Data Output Messages Notifications

CREATE TABLE

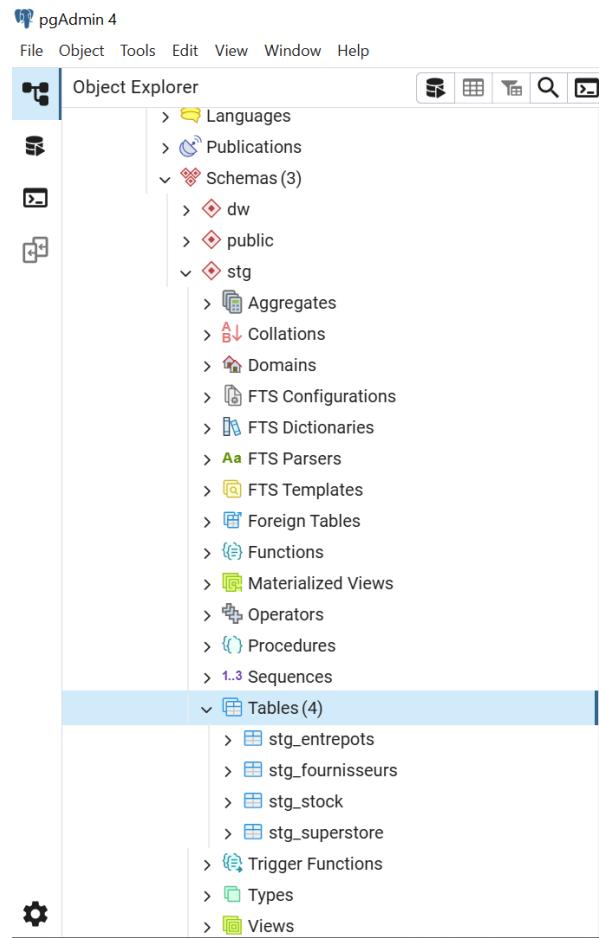
Query returned successfully in 67 msec.

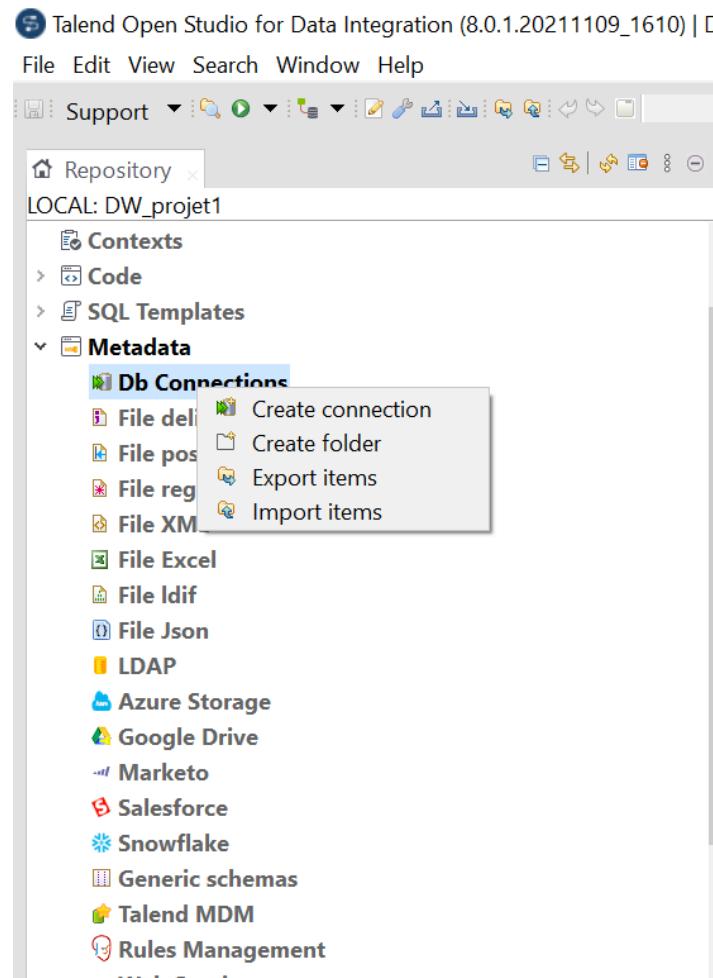
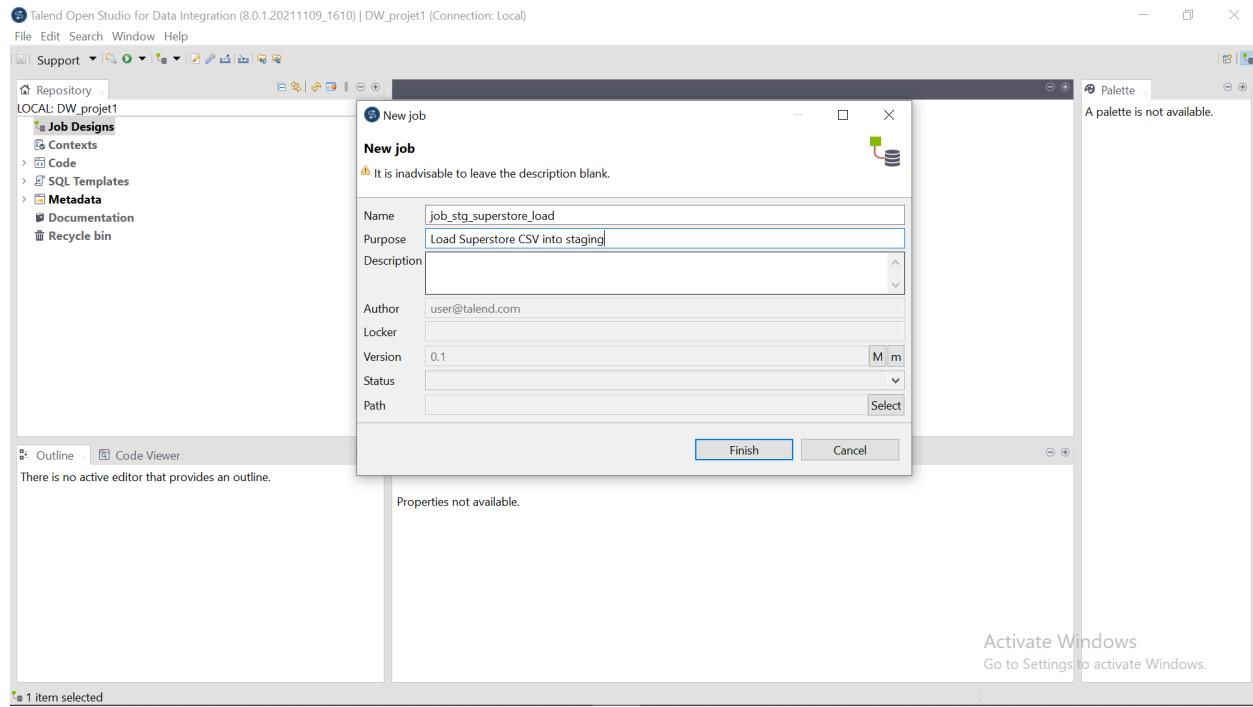
Activate Windows

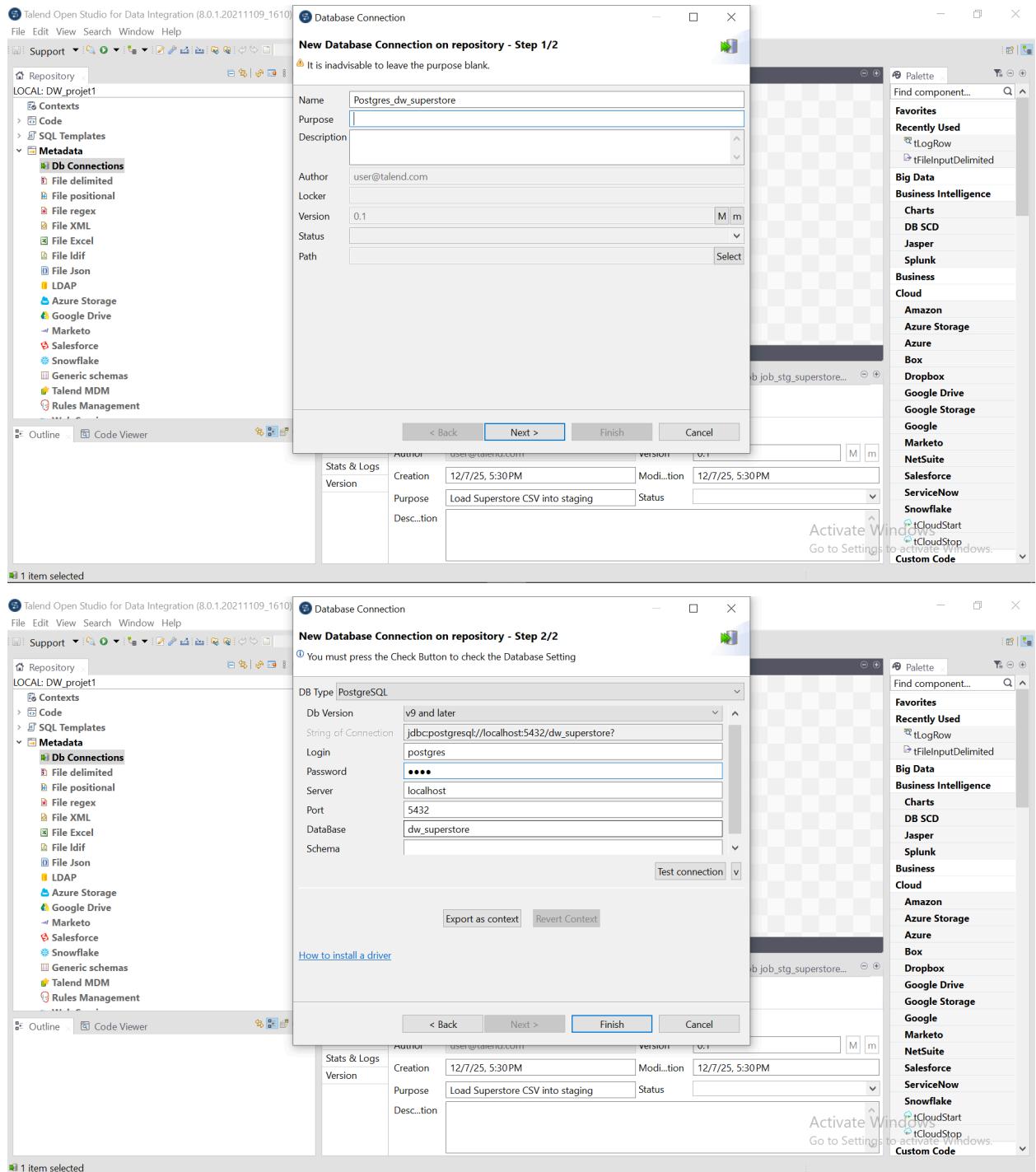
✓ Query returned successfully in 67 msec. X

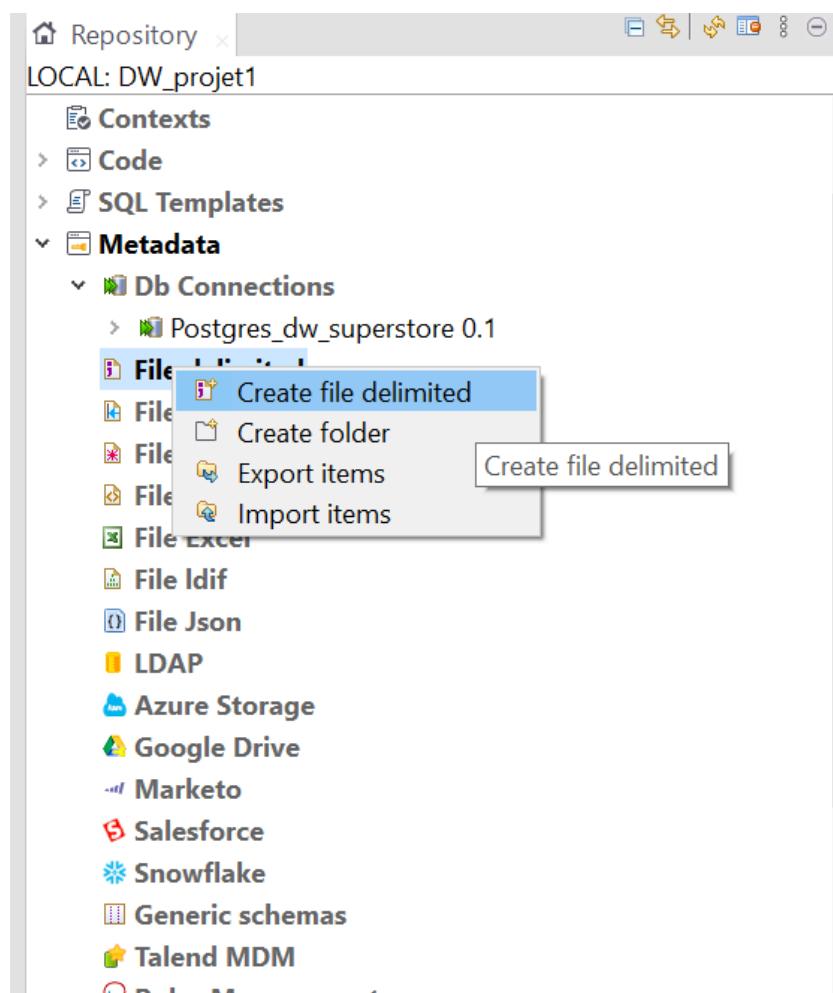
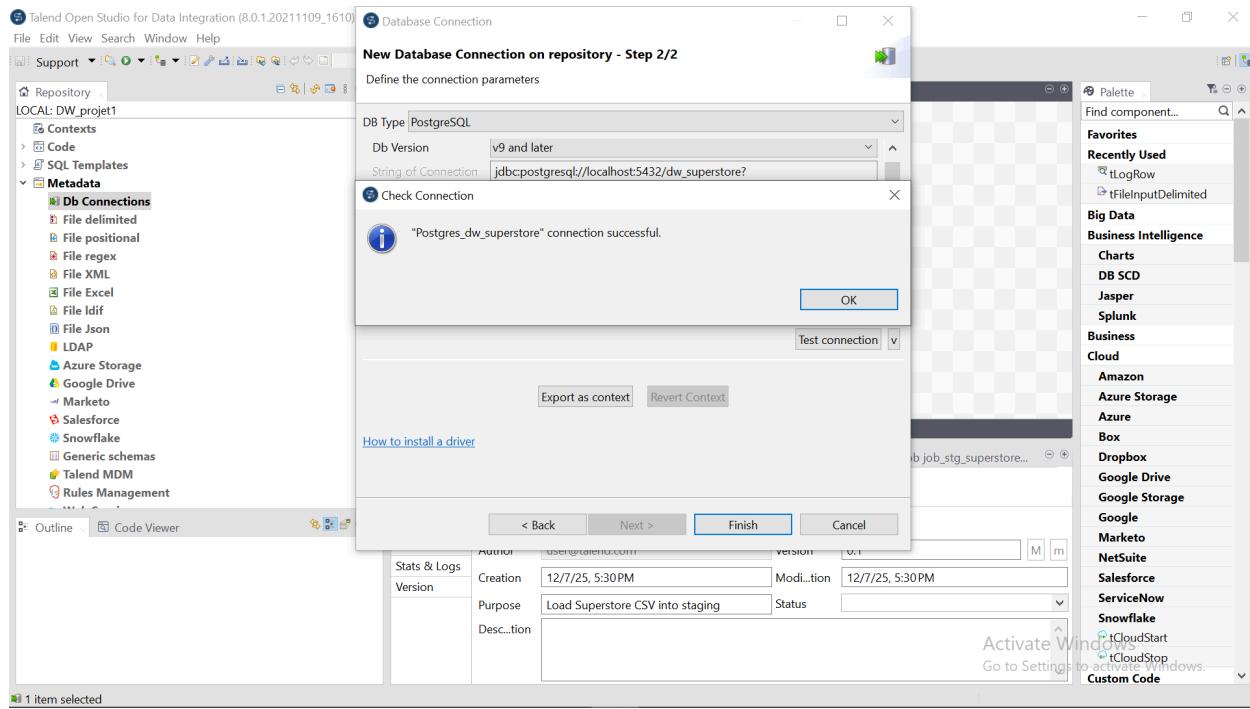
CRLF Ln 9, Col 1

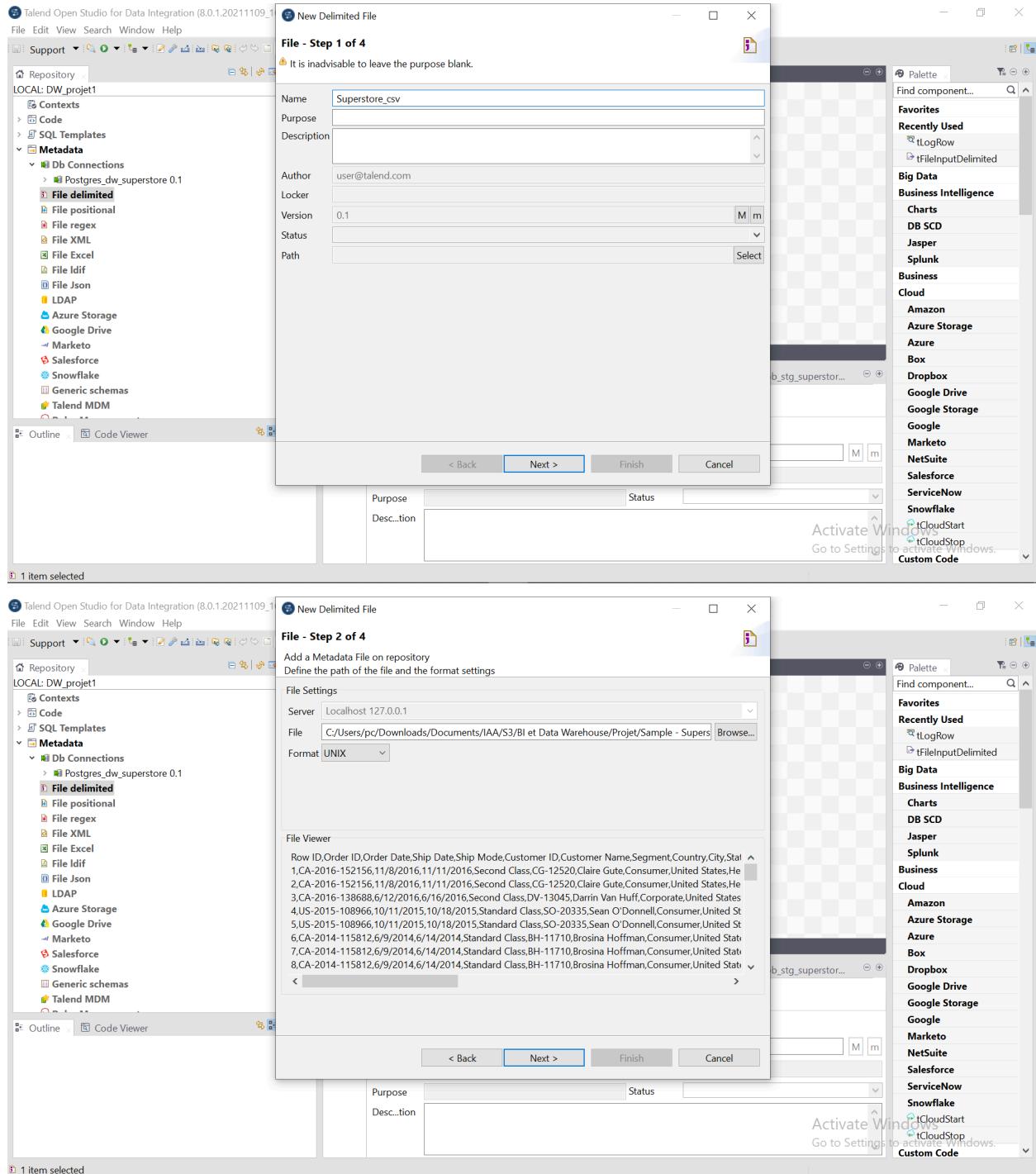
Servers > PostgreSQL 17 > Databases > dw_superstore > Schemas > dw 00:00:00.067











Talend Open Studio for Data Integration (8.0.1.20211109_1)

New Delimited File

File - Step 3 of 4

Add a Metadata File on repository
Define the setting of the parse job

File Settings

- Encoding: UTF-8
- Field Separator: Comma
- Row Separator: Standard
- Escape Char Settings: Delimited
- Text Enclosure: Empty
- Preview/Output: Set heading row as column names

Row ID	Order ID	Order Date	Ship Date	Ship Mode	Customer ID	Customer Name	Segment
1	CA-2016-152156	11/8/2016	11/11/2016	Second Class	CG-12520	Claire Gute	Consumer

Rows To Skip: If any rows must be ignored, specify the following
 Header: 1
 Footer
 Skip empty row
Limit Of Rows: If the number of lines must be limited, specify the following
 Limit: []

Preview/Output: Export as context | Revert Context

< Back | Next > | Finish | Cancel

New Delimited File

File - Step 4 of 4

Add a Schema on repository
Define the Schema

Name: superstore_schema

Comment:

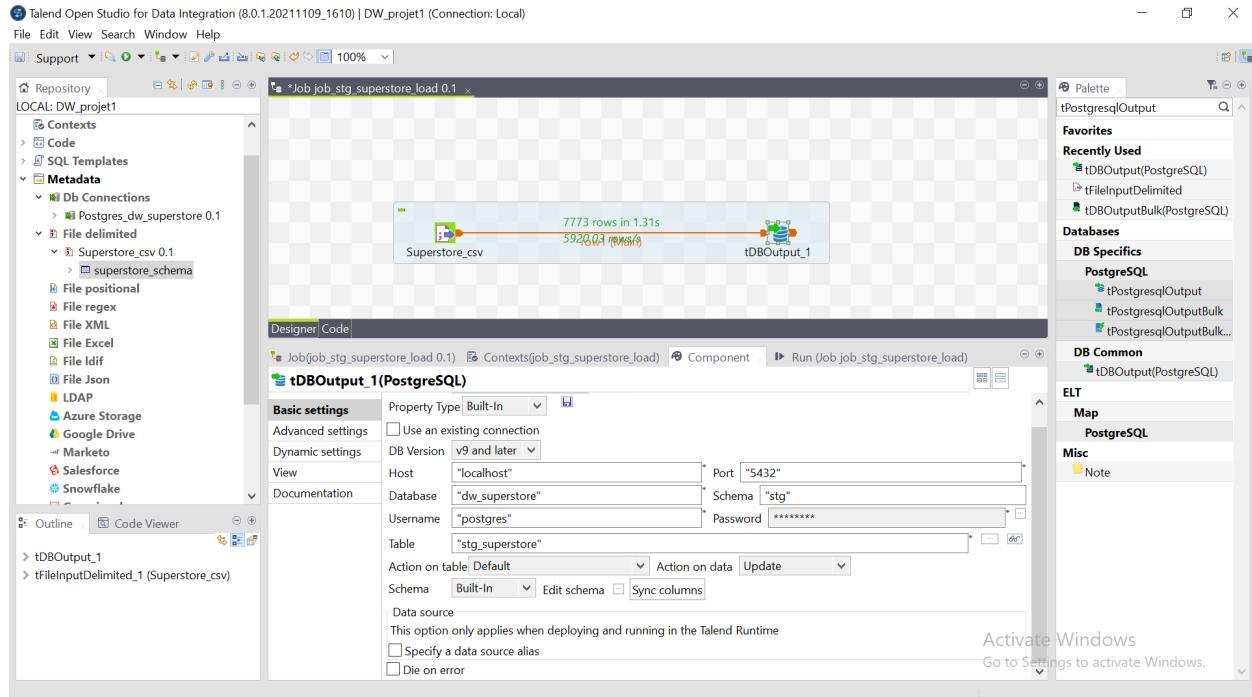
Schema

Click to update schema preview
Guess

Description of the Schema

Column	K..	Type	N..	Date Pattern (...)	Length	Precision	Defa...	Comm...
Row_ID	<input type="checkbox"/>	Integer	<input checked="" type="checkbox"/>		2	0		
Order_ID	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>		14	0		
Order_Date	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>	MM/dd/yyyy...		0		
Ship_Date	<input type="checkbox"/>	Date	<input checked="" type="checkbox"/>	MM/dd/yyyy...		0		
Ship_Mode	<input type="checkbox"/>	String	<input checked="" type="checkbox"/>			0		

< Back | Next > | Finish | Cancel



pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superuser/postgres@PostgreSQL 17*

- public
- stg
 - Aggregates
 - Collations
 - Domains
 - FTS Configuration
 - FTS Dictionaries
 - FTS Parsers
 - FTS Templates
 - Foreign Tables
 - Functions
 - Materialized Views
 - Operators
 - Procedures
 - Sequences
 - Tables (4)
 - stg_entrepot
 - stg_fournisseur
 - stg_stock
 - stg_superstore
 - Trigger Functions
 - Types
 - Views
 - Subscriptions
- postges
- Login/Group Roles

Properties SQL Statistics Dependencies Dependents Processes dw_superuser/postgres@PostgreSQL 17*

Query History

```
1 SELECT COUNT(*) FROM stg.stg_superstore;
2 SELECT * FROM stg.stg_superstore LIMIT 5;
```

Scratch Pad

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

row_id	order_id	order_date	ship_date	ship_mode	customer_id	customer_name	segment	
1	1	CA-2016-152156	2016-11-08	2016-11-11	Second Class	CG-12520	Claire Gute	Consumer
2	3	CA-2016-138688	2016-06-12	2016-06-16	Second Class	DV-13045	Darrin Van Huff	Corporate
3	4	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer
4	5	US-2015-108966	2015-10-11	2015-10-18	Standard Class	SO-20335	Sean O'Donnell	Consumer
5	7	CA-2014-115812	2014-06-09	2014-06-14	Standard Class	BH-11710	Brosina Hoffman	Consumer

Total rows: 5 Query complete 00:00:00.229 Activate Windows Go to Settings to activate Windows.

CRLF Ln 3, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

Properties SQL Statistics Dependencies Dependents Processes

dw_superstore/postgres@PostgreSQL 17 No limit

Query Query History

```
1 SELECT COUNT(*) FROM stg.stg_superstore;
```

Data Output Messages Notifications

	count	bigint
1	15546	

Showing rows: 1 to 1 Page No: 1 of 1

Total rows: 1 Query complete 00:00:00.096

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 2, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

Properties SQL Statistics Dependencies Dependents Processes

dw_superstore/postgres@PostgreSQL 17 No limit

Query Query History

```
1 v INSERT INTO dw.dim_client (
2   customer_id,
3   customer_name,
4   segment,
5   country,
6   region,
7   state,
8   city,
9   postal_code
10 )
11 SELECT DISTINCT
```

Data Output Messages Notifications

INSERT 0 4276

Query returned successfully in 209 msec.

Total rows: 0 Query complete 00:00:00.209

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 23, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

```

category
)
SELECT DISTINCT
product_id,
product_name,
sub_category,
category
FROM stg.stg_superstore
WHERE product_id IS NOT NULL
ORDER BY product_id;

```

Data Output Messages Notifications

INSERT 0 1488

Query returned successfully in 60 msec.

Total rows: Query complete 00:00:00.060 CRLF Ln 15, Col 1

Activate Windows
Go to Settings to activate Windows.

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

```

INSERT INTO dw.dim_produit (
product_id,
product_name,
sub_category,
category
)
SELECT DISTINCT
product_id,
product_name,
sub_category,
category

```

Data Output Messages Notifications

INSERT 0 1488

Query returned successfully in 60 msec.

Total rows: Query complete 00:00:00.060 CRLF Ln 5, Col 13

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

Properties SQL Statistics Dependencies Dependents Processes

dw_superstore/postgres@PostgreSQL 17

No limit

Query Query History

```

1 ✓ INSERT INTO dw.dim_ship_mode (ship_mode_name)
2   SELECT DISTINCT
3     ship_mode
4   FROM stg.stg_superstore
5   WHERE ship_mode IS NOT NULL
6   ORDER BY ship_mode;
7

```

Data Output Messages Notifications

INSERT 0 4

Query returned successfully in 68 msec.

Total rows: Query complete 00:00:00.068

Activate Windows

✓ Query returned successfully in 68 msec. X

CRLF Ln 7, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer dw_superstore/postgres@PostgreSQL 17*

Properties SQL Statistics Dependencies Dependents Processes

dw_superstore/postgres@PostgreSQL 17

No limit

Query Query History

```

1 ✓ WITH bounds AS (
2   SELECT
3     LEAST(MIN(order_date), MIN(ship_date)) AS min_date,
4     GREATEST(MAX(order_date), MAX(ship_date)) AS max_date
5   FROM stg.stg_superstore
6 ),
7   dates AS (
8   SELECT generate_series(min_date, max_date, interval '1 day')::date AS d
9   FROM bounds
10 )
11 INSERT INTO dw.dim_date (

```

Data Output Messages Notifications

INSERT 0 1463

Query returned successfully in 67 msec.

Total rows: Query complete 00:00:00.067

Activate Windows

✓ Query returned successfully in 67 msec. X

CRLF Ln 36, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

dw_superstore/postgres@PostgreSQL 17 No limit Scratch Pad

Query Query History

```

17     p.product_key,
18     s.ship_mode_key,
19     sst.order_id,
20     sst.sales,
21     sst.quantity,
22     sst.discount,
23     sst.profit
24   FROM stg.stg_superstore sst
25   JOIN dw.dim_client c  ON c.customer_id      = sst.customer_id
26   JOIN dw.dim_produit p  ON p.product_id      = sst.product_id
27   JOIN dw.dim_ship_mode s  ON s.ship_mode_name = sst.ship_mode
28   JOIN dw.dim_date d_cmd ON d_cmd.full_date   = sst.order_date
29   JOIN dw.dim_date d_ship ON d_ship.full_date = sst.ship_date;
30

```

Data Output Messages Notifications

INSERT 0 101846

Query returned successfully in 5 secs 36 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: Query complete 00:00:05.036 CRLF Ln 30, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Properties SQL Statistics Dependencies Dependents Processes dw_superstore/postgres@PostgreSQL 17*

dw_superstore/postgres@PostgreSQL 17 No limit Scratch Pad

Query Query History

```

1  SELECT
2     f.order_id,
3     d_cmd.full_date AS order_date,
4     c.customer_name,
5     p.product_name,
6     f.sales,
7     f.quantity,
8     f.profit
9   FROM dw.fact_vente f
10  JOIN dw.dim_client c  ON c.client_key      = f.client_key
11  JOIN dw.dim_produit p  ON p.product_key    = f.product_key
12  JOIN dw.dim_date d_cmd ON d_cmd.date_key   = f.date_commande_key
13
14  LIMIT 10;

```

Data Output Messages Notifications

order_id	order_date	customer_name	product_name	sales	quantity	profit
CA-2016-139688	2016-06-12	Darrin Van Huff	Self-Adhesive Address Labels for Typewriters by Universal	14.62	2	
US-2015-108966	2015-10-11	Sean O'Donnell	Bretford CR4500 Series Slim Rectangular Table	957.58	5	
US-2015-108966	2015-10-11	Sean O'Donnell	Bretford CR4500 Series Slim Rectangular Table	957.58	5	
US-2015-108966	2015-10-11	Sean O'Donnell	Bretford CR4500 Series Slim Rectangular Table	957.58	5	
US-2015-108966	2015-10-11	Sean O'Donnell	Bretford CR4500 Series Slim Rectangular Table	957.58	5	

Showing rows: 1 to 10 Page No: 1 of 1

Activate Windows
Go to Settings to activate Windows.

Total rows: 10 Query complete 00:00:00.092 CRLF Ln 14, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

dw_superstore/postgres@PostgreSQL 17*

Query **Query History**

```

1 SELECT
2     d.annee,
3     c.region,
4     SUM(f.sales) AS total_sales,
5     SUM(f.profit) AS total_profit
6 FROM dw.fact_vente f
7 JOIN dw.dim_date d ON d.date_key = f.date_commande_key
8 JOIN dw.dim_client c ON c.client_key = f.client_key
9 GROUP BY ROLLUP (d.annee, c.region)
10 ORDER BY d.annee, c.region;

```

Data Output **Messages** **Notifications**

	annee	region	total_sales	total_profit
1	2014	Central	1248284.74	133156.96
2	2014	East	1426157.86	100838.98
3	2014	South	961408.18	112474.98
4	2014	West	1727905.68	248197.54
5	2014	[null]	5363756.46	594668.46
6	2015	Central	1242897.00	197801.94
7	2015	East	1347001.72	220050.96
8	2015	South	844770.88	108138.24

Total rows: 21 Query complete 00:00:00.128 CRLF Ln 11, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

dw_superstore/postgres@PostgreSQL 17*

Query **Query History**

```

1 INSERT INTO dw.dim_fournisseur (
2     fournisseur_id,
3     nom_fournisseur,
4     pays,
5     email_contact
6 )
7 SELECT DISTINCT
8     fournisseur_id,
9     nom_fournisseur,
10    pays,
11    email_contact
12 FROM stg.stg_fournisseurs
13 WHERE fournisseur_id IS NOT NULL
14 ORDER BY fournisseur_id;
15

```

Data Output **Messages** **Notifications**

INSERT 0 0

Query returned successfully in 93 msec.

Activate Windows
Go to Settings to activate Windows.

Total rows: 0 Query complete 00:00:00.093 CRLF Ln 15, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Card X Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17* X

dw_superstore/postgres@PostgreSQL 17*

Query History Execute script F5 Scratch Pad

```

1 v INSERT INTO dw.dim_entrepot (
2     entrepot_id,
3     nom_entrepot,
4     region
5 )
6 SELECT DISTINCT
7     entrepot_id,
8     nom_entrepot,
9     region
10 FROM stg.stg_entrepots
11 WHERE entrepot_id IS NOT NULL
12 ORDER BY entrepot_id;
13

```

Data Output Messages Notifications

INSERT 0 0

Query returned successfully in 183 msec.

Total rows: Query complete 00:00:00.183

Activate Windows
Go to Settings to activate Windows.

✓ Query returned successfully in 183 msec. X

CRLF Ln 13, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Card X Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17* X

dw_superstore/postgres@PostgreSQL 17*

Query History Execute script F5 Scratch Pad

```

1 v INSERT INTO dw.fact_stock (
2     date_stock_key,
3     produit_key,
4     entrepot_key,
5     fournisseur_key,
6     quantite_stock
7 )
8 SELECT
9     d.date_key AS date_stock_key,
10    p.produit_key,
11    e.entrepot_key,
12    f.fournisseur_key,
13    sst.quantite_stock
14 FROM stg.stg_stock sst
15 JOIN dw.dim_produit p ON p.product_id = sst.product_id
16 JOIN dw.dim_entrepot e ON e.entrepot_id = sst.entrepot_id
17 JOIN dw.dim_fournisseur f ON f.fournisseur_id = sst.fournisseur_id

```

Data Output Messages Notifications

INSERT 0 0

Query returned successfully in 105 msec.

Total rows: Query complete 00:00:00.105

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 19, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Card X Properties X SQL X Statistics X Dependencies X Dependents X Processes X dw_superstore/postgres@PostgreSQL 17* X :

dw_superstore/postgres@PostgreSQL 17* Scratch Pad X

Query _ Query History

```
1 SELECT COUNT(*) FROM dw.dim_entrepot;
2 SELECT COUNT(*) FROM dw.dim_fournisseur;
3 SELECT COUNT(*) FROM dw.fact_stock;
4 |
```

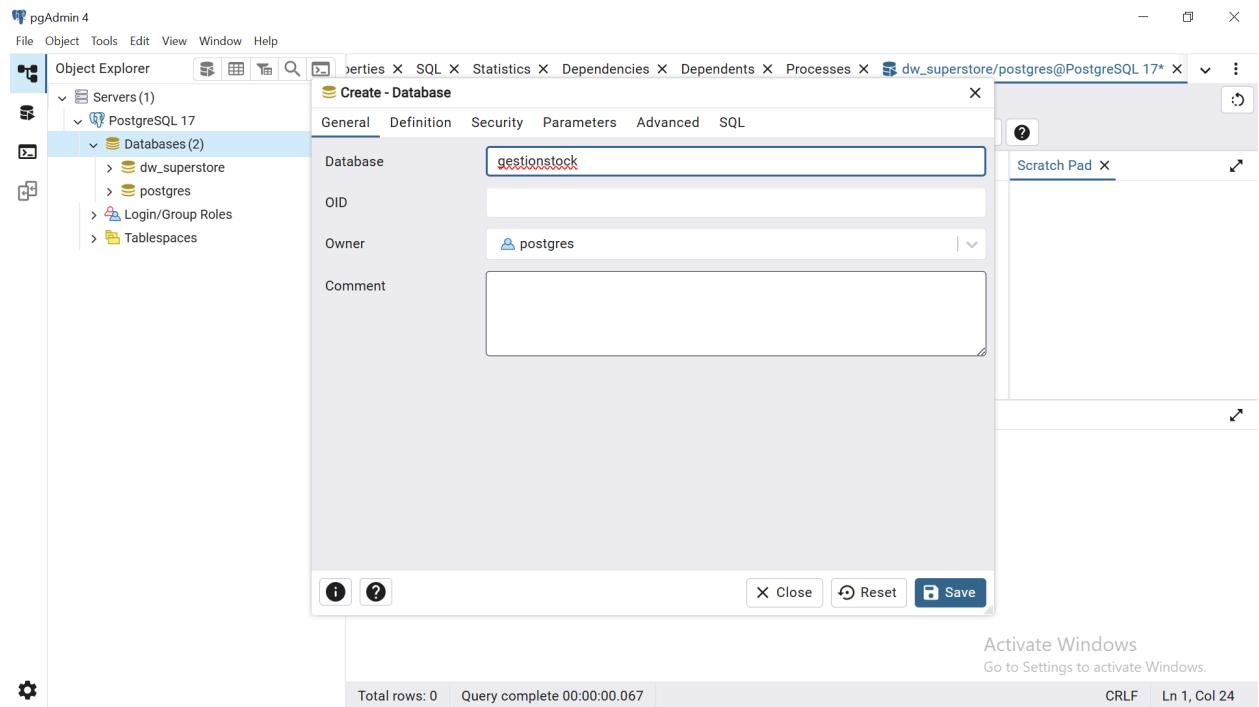
Data Output Messages Notifications

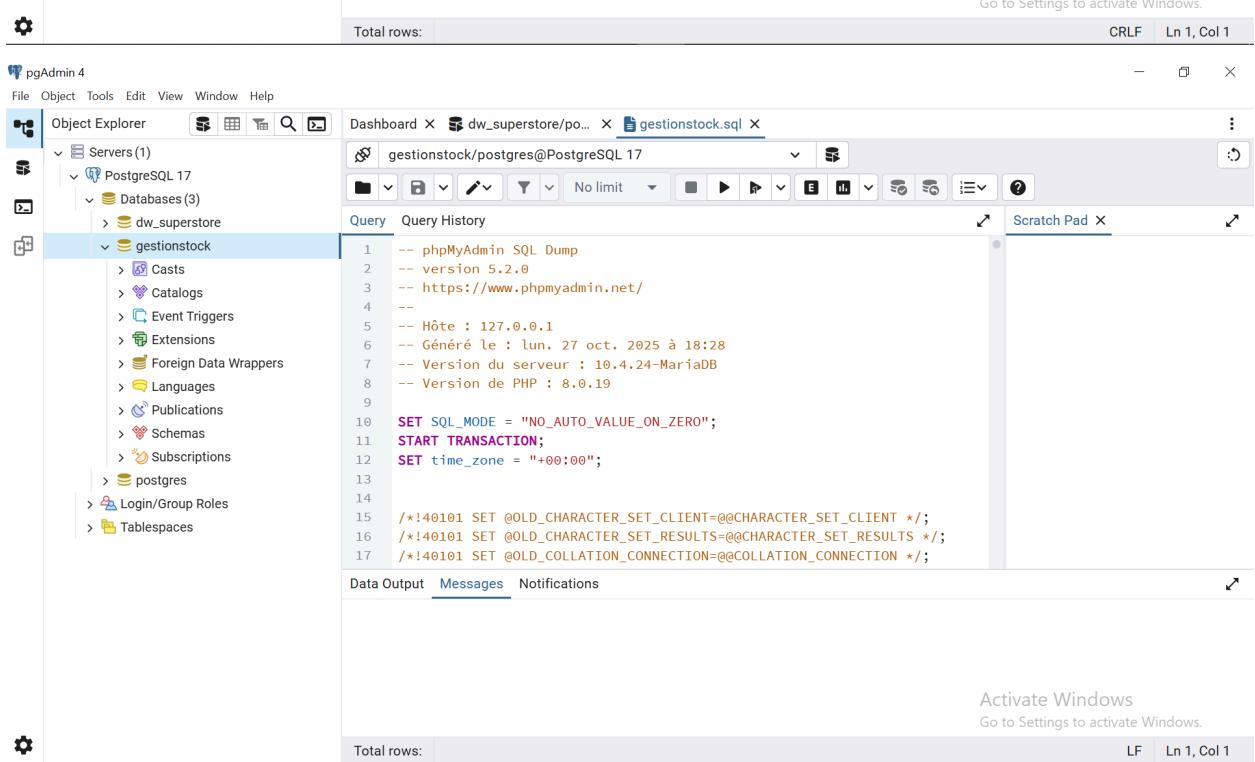
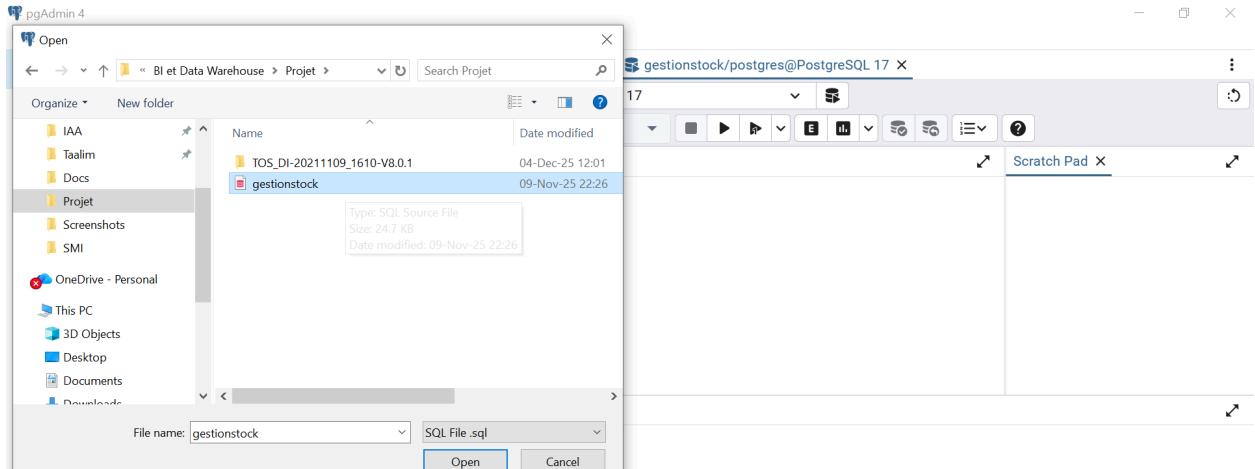
	count	bigint
1	0	

Showing rows: 1 to 1 Page No: 1 of 1 CRLF Ln 4, Col 1

Total rows: 1 Query complete 00:00:00.154

Activate Windows
Go to Settings to activate Windows.





pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > Publications
- < Schemas (1)
 - public
- > Aggregates
- > Collations
- > Domains
- > FTS Configurations
- > FTS Dictionaries
- > FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- < Tables (3)
 - entrepots
 - fournisseurs
 - stock
- > Trigger Functions
- > Types
- > Views
- > Subscriptions
- > postgres
- > Login/Group Roles
- > Tablespaces

Dashboard > dw_superstore/postgres@PostgreSQL 17 > gestionstock.sql*

Query Query History

```

1 -- =====
2 -- PostgreSQL version of gestionstock dump
3 --
4
5 BEGIN;
6
7 -- 1. Drop old tables if they exist (optional)
8 DROP TABLE IF EXISTS stock;
9 DROP TABLE IF EXISTS fournisseurs;
10 DROP TABLE IF EXISTS entrepots;
11
12 -- 2. Tables definition
13 -----
14 Table: entrepots

```

Data Output Messages Notifications

NOTICE: table "stock" does not exist, skipping
 NOTICE: table "fournisseurs" does not exist, skipping
 NOTICE: table "entrepots" does not exist, skipping
 COMMIT

Query returned successfully in 84 msec.

Activate Windows
 Go to Settings to activate Windows.

Total rows: 0 Query complete 00:00:00.084 LF Ln 7, Col 27

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer

- > Publications
- < Schemas (1)
 - public
- > Aggregates
- > Collations
- > Domains
- > FTS Configurations
- > FTS Dictionaries
- > FTS Parsers
- > FTS Templates
- > Foreign Tables
- > Functions
- > Materialized Views
- > Operators
- > Procedures
- > Sequences
- < Tables
 - entrepot
 - fournisseur
 - stock
- > Trigger Functions
- > Types
- > Views
- > Subscriptions
- > postgres
- > Login/Group Roles
- > Tablespaces

Dashboard > dw_superstore/postgres@PostgreSQL 17 > gestionstock.sql*

Query Query History

```

1 SELECT * FROM entrepots LIMIT 5;
2 SELECT * FROM fournisseurs LIMIT 5;
3 SELECT * FROM stock LIMIT 5;
4

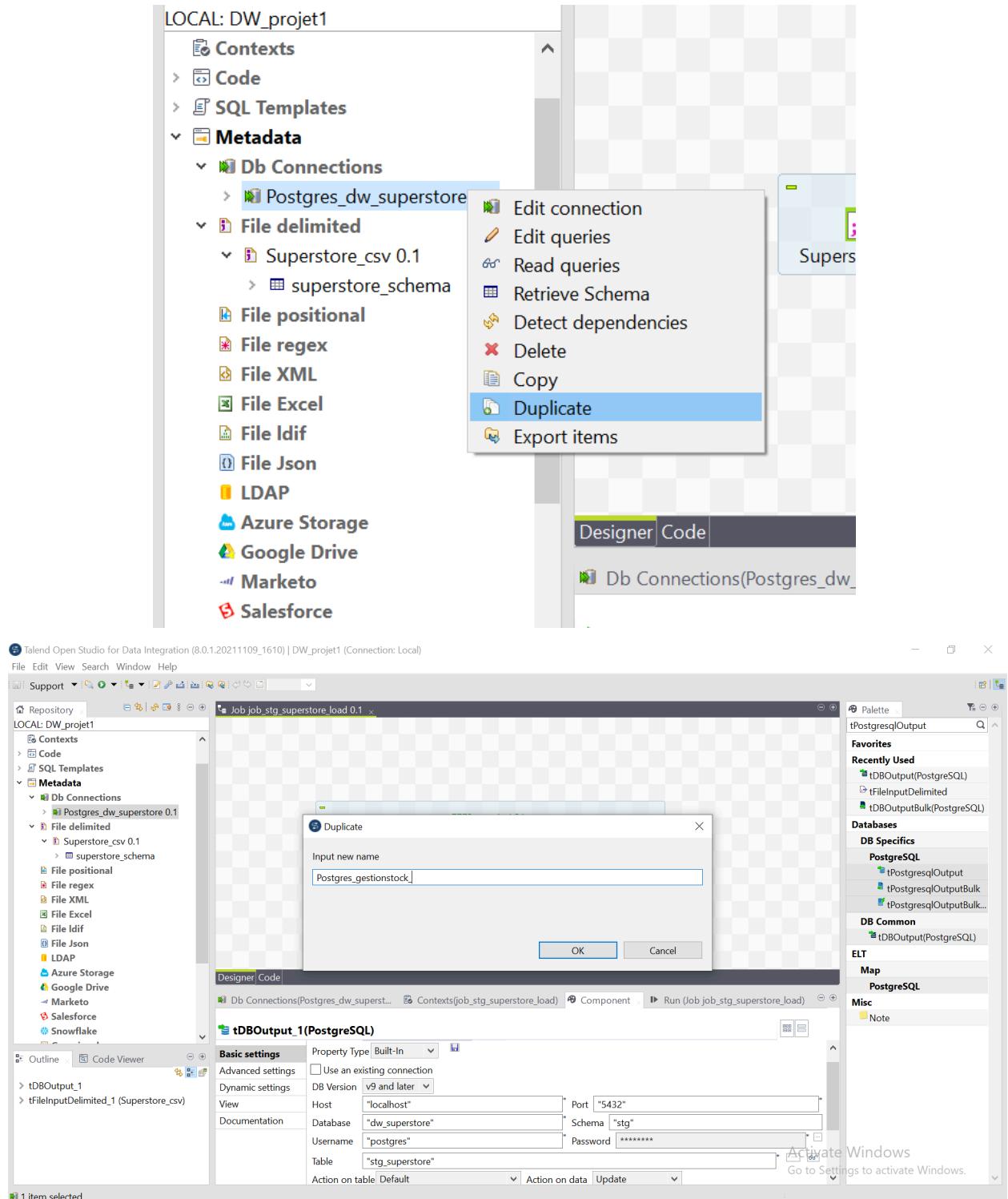
```

Data Output Messages Notifications

	product_id	nom_produit	fournisseur_id	entrepot_id	quantite_stock	date_dernier_reapprovisionnement
1	FUR-BO-10001798	Bush Somerset Collection Bookcase	3	5	45	2024-01-15
2	FUR-BO-10002545	Atlantic Metals Mobile 3-Shelf Bookcases, Custom Colors	3	5	15	2024-02-17
3	FUR-BO-10002613	Atlantic Metals Mobile 4-Shelf Bookcases, Custom Colors	3	5	11	2024-06-03
4	FUR-BO-10004695	O'Sullivan 2-Door Barrister Bookcase in Odessa Pine	3	5	7	2024-04-23
5	FUR-BO-10004834	Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish	3	5	8	2024-03-22

Activate Windows
 Go to Settings to activate Windows.

Total rows: 5 Query complete 00:00:00.086 LF Ln 4, Col 1



- Edit connection
- Edit queries
- Read queries
- Retrieve Schema
- Detect dependencies
- Delete
- Copy
- Duplicate**
- Export items

Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW_projet1 (Connection: Local)

File Edit View Search Window Help

Support | Repository | Contexts | SQL Templates | Metadata | Db Connections | File delimited | File positional | File regex | File XML | File Excel | File Idif | File Json | LDAP | Azure Storage | Google Drive | Marketo | Salesforce | Snowflake

Job job_stg_superstore_load 0.1

Duplicate

Input new name
Postgres_gestionstock_

OK Cancel

tDBOOutput_1 (PostgreSQL)

Basic settings

Property Type: Built-In

Advanced settings

Dynamic settings

View

Documentation

Host: "localhost" Port: "5432"

Database: "dw_superstore" Schema: "stg"

Username: "postgres" Password: "*****"

Table: "stg_superstore"

Action on table: Default Action on data: Update

Palette tPostgresOutput

Favorites

Recently Used

Databases

DB Specifics

PostgreSQL

DB Common

ELT

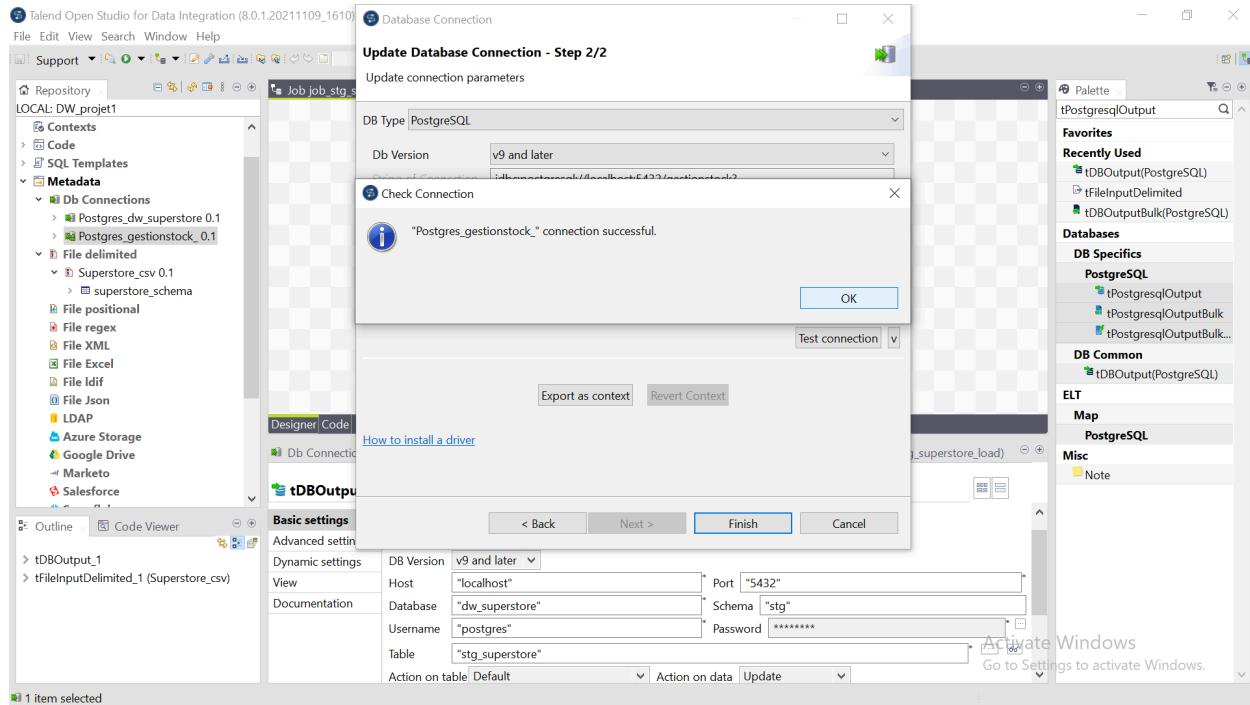
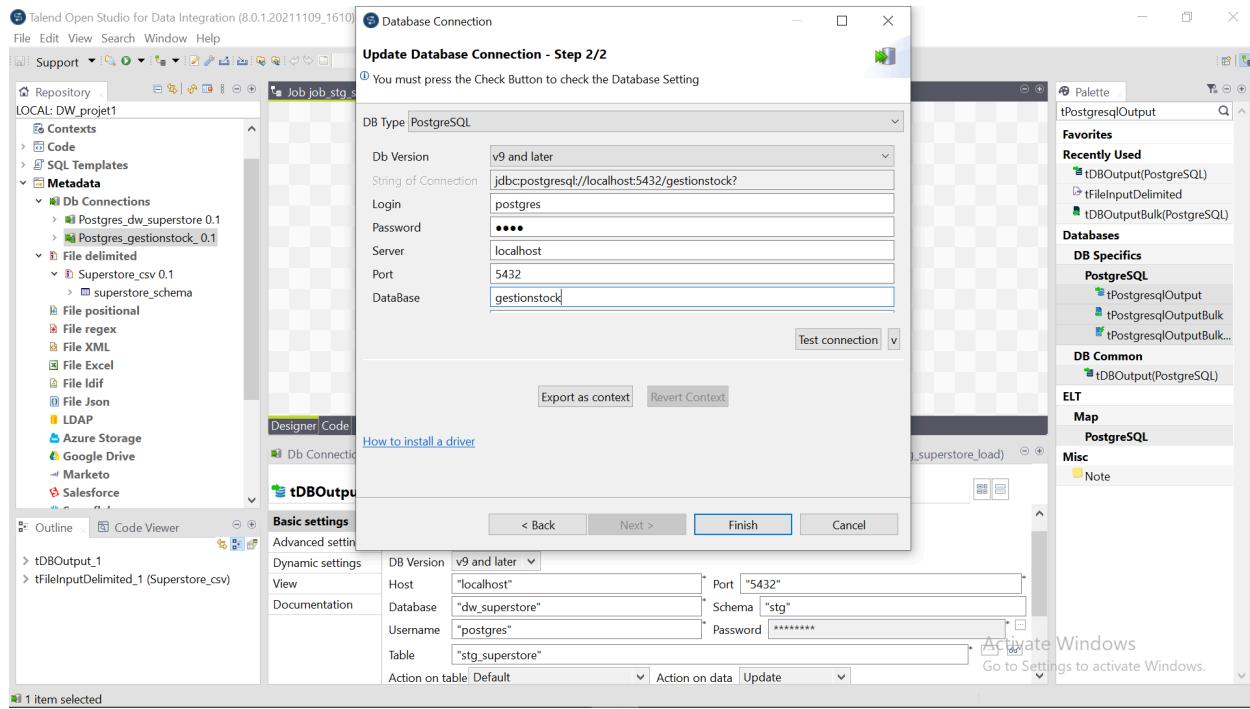
Map

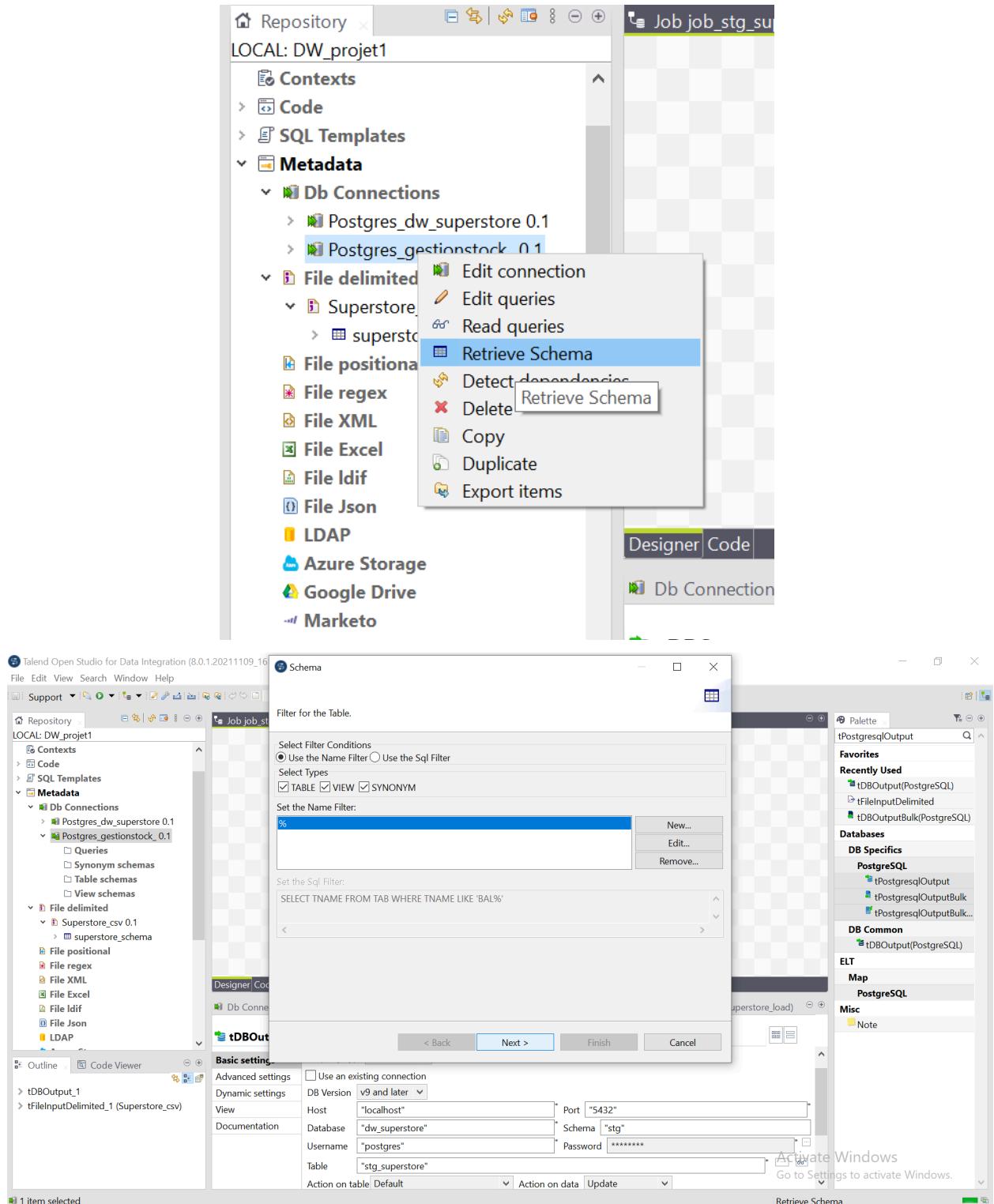
PostgreSQL

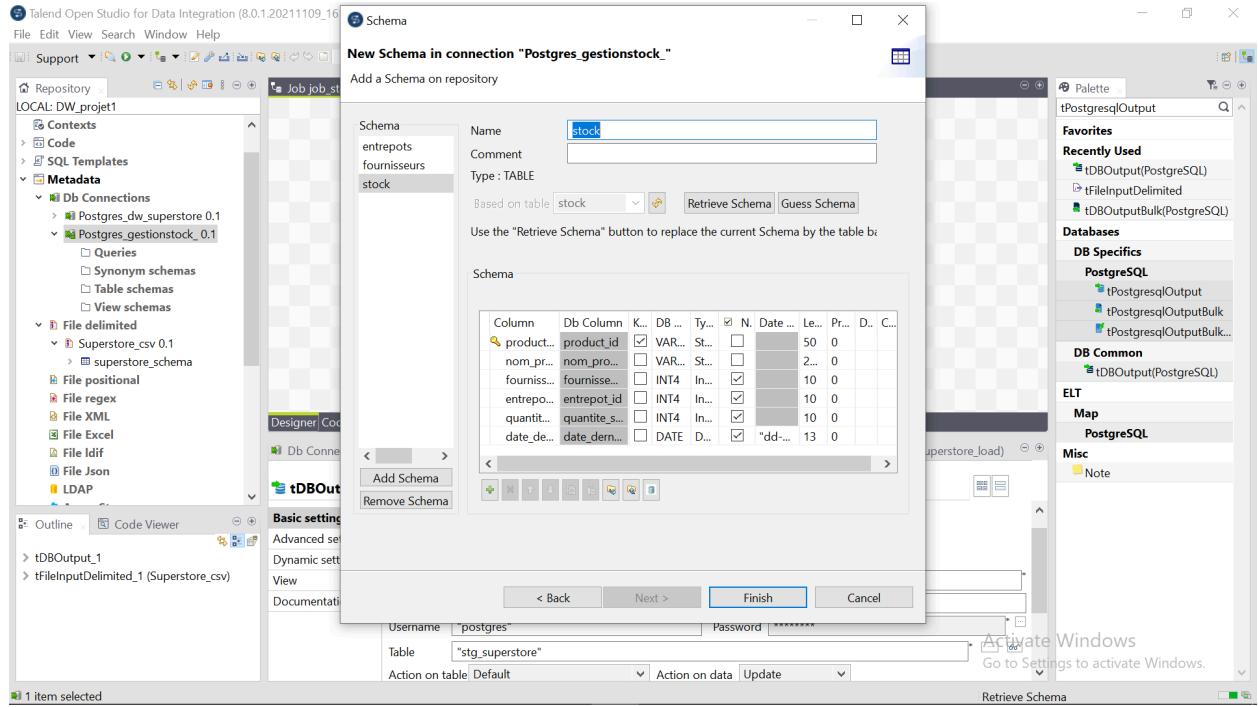
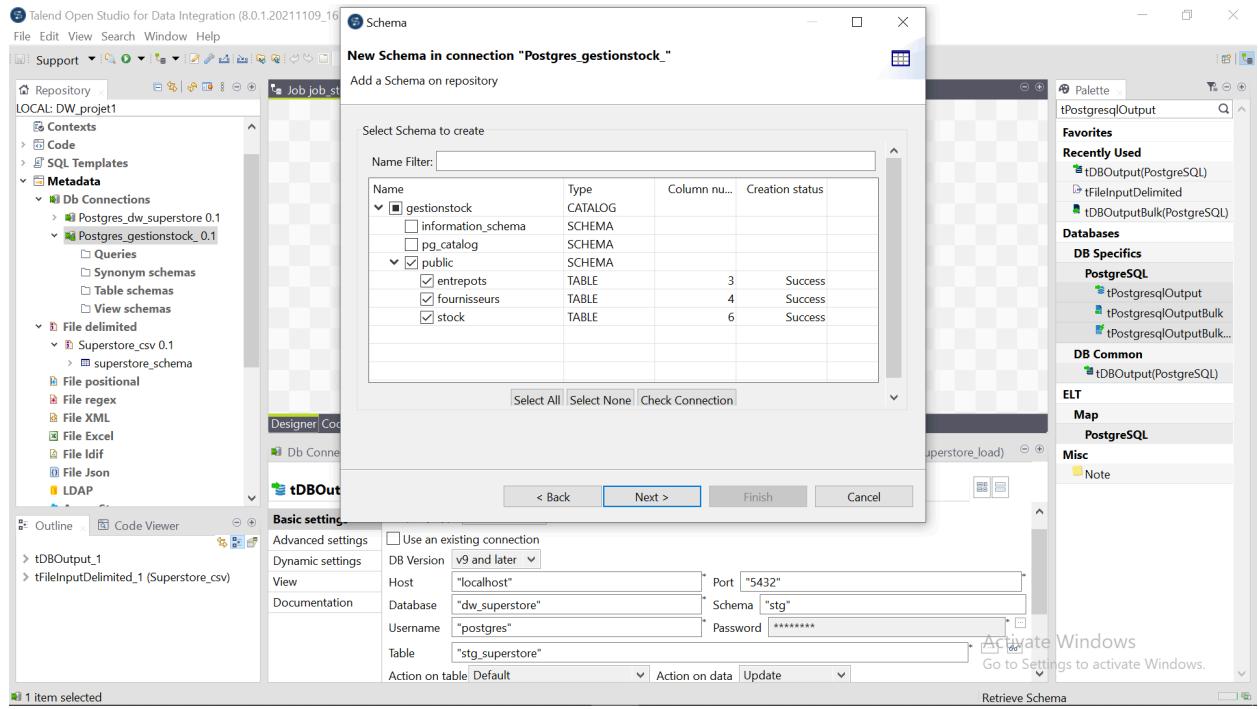
Misc

Note

Activate Windows
Go to Settings to activate Windows.







Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW_projet1 (Connection: Local)

New job

Name: job_stg_entrepots_load

Purpose:

Description:

Author: user@talend.com

Locker:

Version: 0.1

Status:

Path:

tDBOutput_1 (PostgreSQL)

Property Type: Built-In

Advanced settings

Dynamic settings

View

Documentation

Host: "localhost"

Port: "5432"

Database: "dw_superstore"

Schema: "stg"

Username: "postgres"

Password: *****

Action on table: Default

Action on data: Update

Schema

New Schema in connection "Postgres_dw_superstore"

Add a Schema on repository

Select Schema to create

Name Filter:

Name	Type	Column nu...	Creation status
dw_superstore	CATALOG		
dw	SCHEMA		
information_schema	SCHEMA		
pg_catalog	SCHEMA		
public	SCHEMA		
stg	SCHEMA		

Select All | Select None | Check Connection

< Back | Next > | Finish | Cancel

Palette

tPostgresqlOutput

Favorites

Recently Used

- tDBOutput(PostgreSQL)
- tFileInputDelimited
- tDBOutputBulk(PostgreSQL)

Databases

DB Specifics

- PostgreSQL
- tPostgresqlOutput
- tPostgresqlOutputBulk
- tPostgresqlOutputBulk...

DB Common

- tDBOutput(PostgreSQL)

ELT

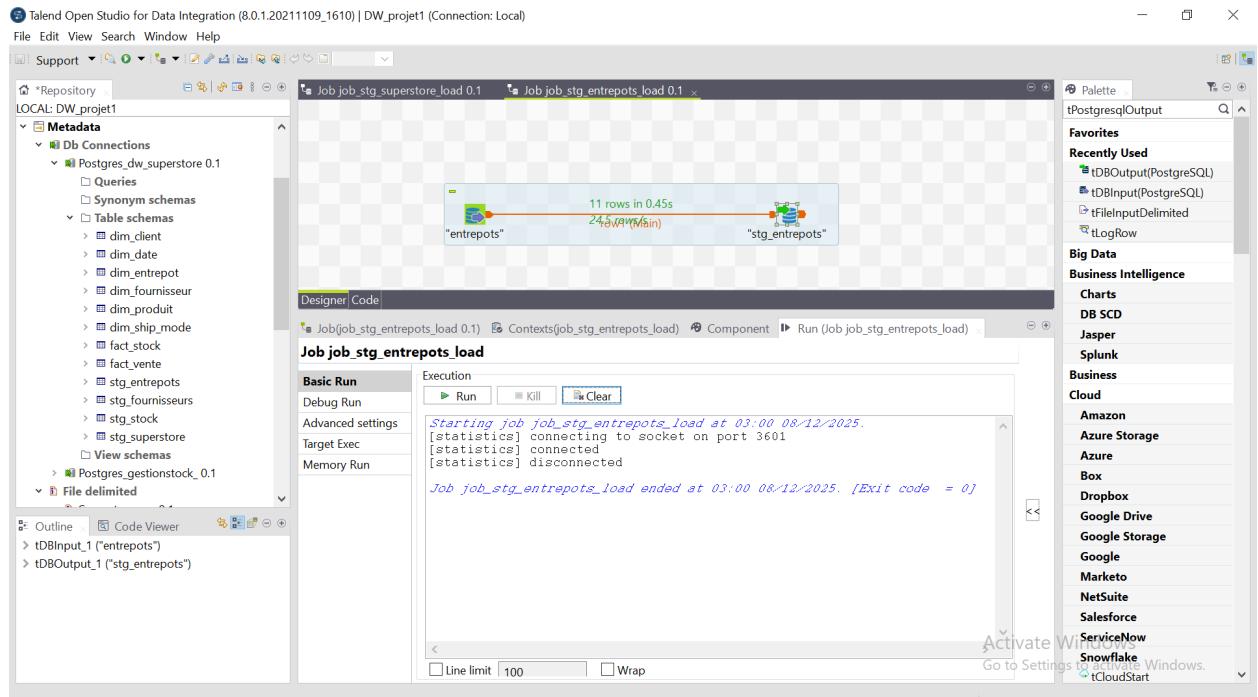
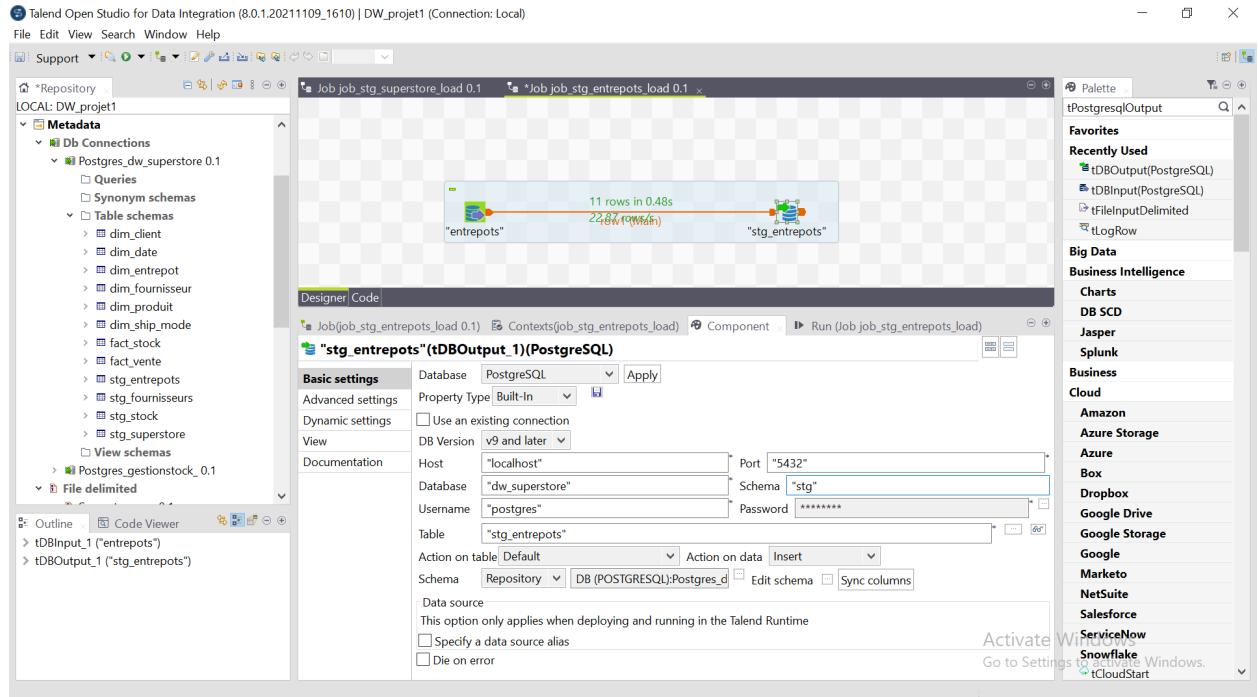
Map PostgreSQL

Misc Note

Activate Windows Go to Settings to activate Windows.

1 item selected

1 item selected



pgAdmin 4

File Object Tools Edit View Window Help

Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock.sql*

Query Query History

```
1 SELECT COUNT(*) FROM stg.stg_entrepots;
2 SELECT * FROM stg.stg_entrepots LIMIT 5;
```

Data Output Messages Notifications

Showing rows: 1 to 5 Page No: 1 of 1

	entrepot_id [PK] integer	nom_entrepot character varying (100)	region character varying (100)
1	1	Entrepot Ouest	Ouest
2	2	Entrepot Est	Est
3	3	Entrepot Sud	Sud
4	4	Entrepot Nord	Nord
5	5	Entrepot Centre	Centre

Total rows: 5 Query complete 00:00:00.115 CRLF Ln 3, Col 1

Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW_projet1 (Connection: Local)

File Edit View Search Window Help

Support Repository LOCAL: DW_projet1

Job Designs job_stg_entrepots_load 0.1 job_stg_superstore_load 0.1

New job

It is inadvisable to leave the purpose blank.

Name: job_stg_fournisseurs_load

Purpose:

Description:

Author: user@talend.com

Locker:

Version: 0.1

Status:

Path:

Finish Cancel

Designer

Basic settings

Advanced

Dynamic

View

Document

tPostgresOutput

Favorites

Recently Used

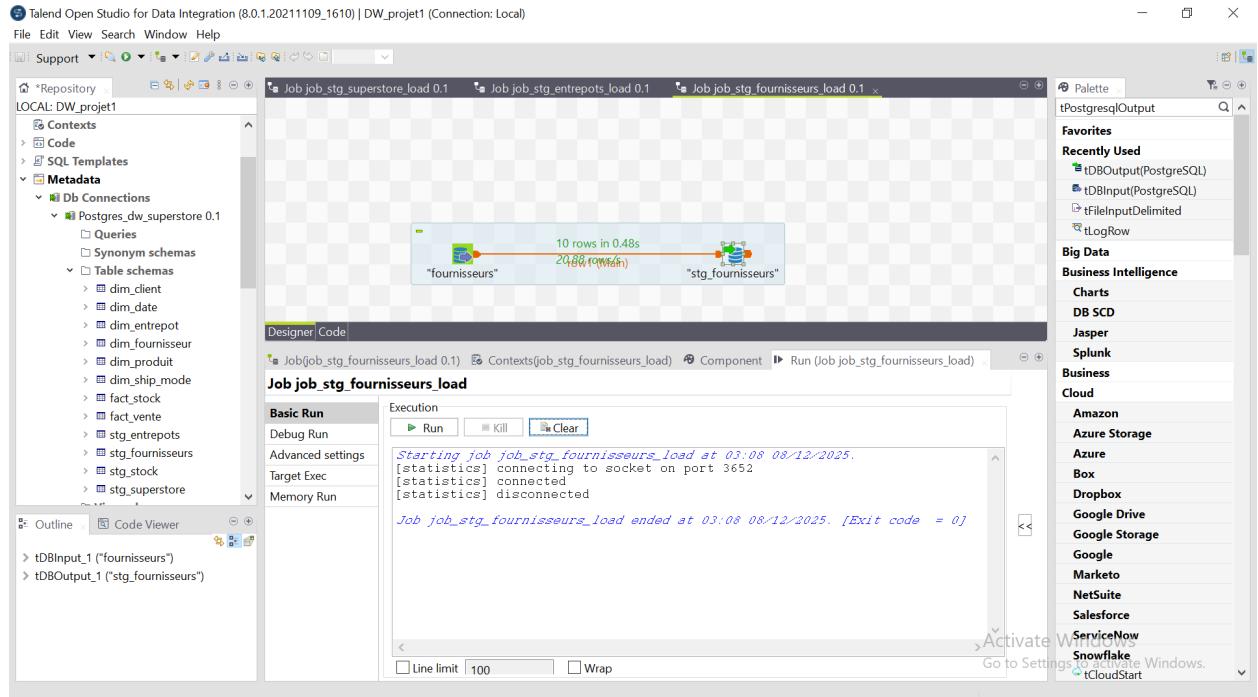
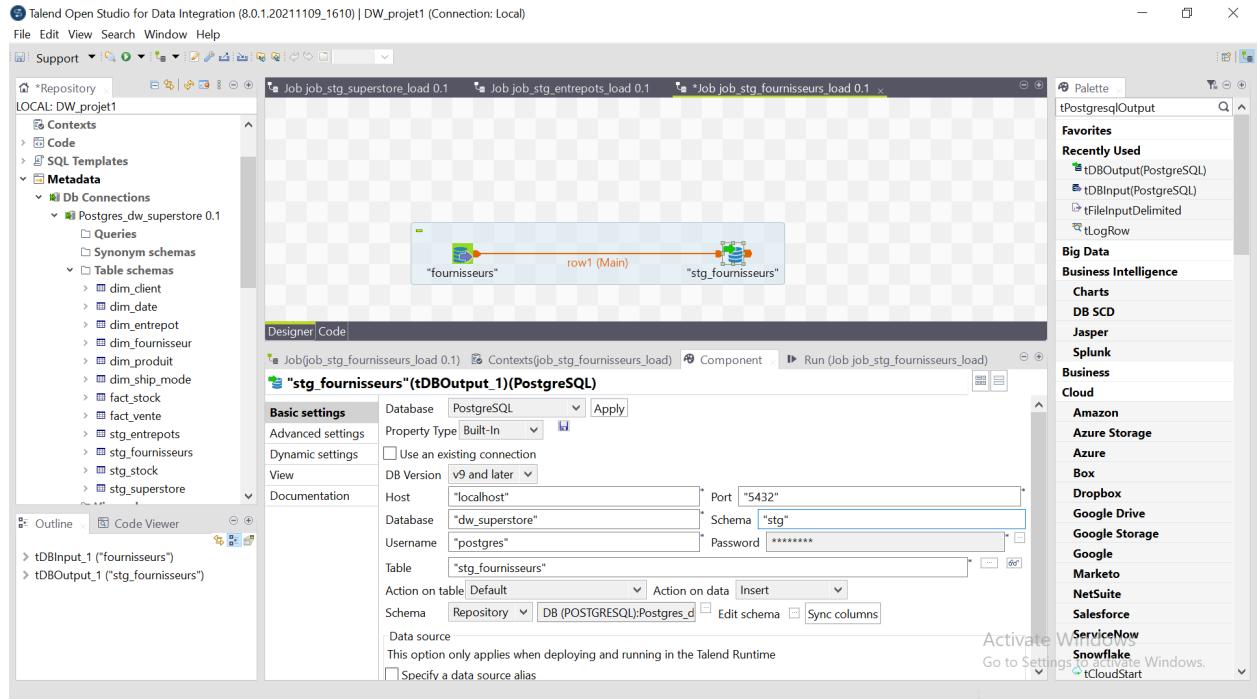
- tDBOutput(PostgreSQL)
- tDBInput(PostgreSQL)
- tFileInputDelimited
- tLogRow

Big Data

Business Intelligence

- Charts
- DB SCD
- Jasper
- Splunk
- Business
- Cloud
- Amazon
- Azure Storage
- Azure
- Box
- Dropbox
- Google Drive
- Google Storage
- Google
- Marketo
- NetSuite
- Salesforce
- ServiceNow
- Snowflake
- tCloudStart

Activate Windows
Go to Settings to activate Windows.



pgAdmin 4

File Object Tools Edit View Window Help

Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock.sql*

Query Query History

```
1 SELECT COUNT(*) FROM stg.stg_fournisseurs;
2 SELECT * FROM stg.stg_fournisseurs LIMIT 5;
```

Data Output Messages Notifications

Tables

fournisseur_id [PK] integer	nom_fournisseur character varying (150)	pays character varying (100)	email_contact character varying (150)
1	Fournitures USA	États-Unis	contact@fournituresusa.com
2	Mobilier Europe	France	contact@mobilierfrance.fr
3	Papeterie Monde	Allemagne	contact@papeteriemonde.de
4	Techno Supplies	Royaume-Uni	contact@technosupplies.co.uk
5	Bureautique Inc.	Canada	contact@bureautiqueinc.ca

Showing rows: 1 to 5 Page No: 1 of 1

Total rows: 5 Query complete 00:00:00.094

Activate Windows

Successfully run. Total query runtime: 94 msec. 5 rows affected.

CRLF Ln 3, Col 1

Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW_projet1 (Connection: Local)

File Edit View Search Window Help

Support

Repository LOCAL: DW_projet1

Job Designs

- job_stg_entrepots_load 0.1
- job_stg_fournisseurs_load 0.1
- job_stg_superstore_load 0.1

Contexts

Code

SQL Templates

Metadata

- Db Connections
- Postgres_dw_superstore 0.1
- Queries
- Synonym schemas
- Table schemas
- dim_client
- dim_date
- dim_entrepot
- dim_fournisseur
- dim_produit
- dim_ship_mode
- fact_stock
- fact_vente

Outline

Code Viewer

New job

New job

It is inadvisable to leave the purpose blank.

Name: job_stg_stock_load

Purpose:

Description:

Author: user@talend.com

Locker:

Version: 0.1

Status:

Path:

Finish Cancel

[statistics] connected
[statistics] disconnected

Job job_stg_fournisseurs_load ended at 03:08 08/12/2025. [Exit code = 0]

tPostgresqlOutput

Favorites

Recently Used

- tDBOutput(PostgreSQL)
- tDBInput(PostgreSQL)
- tFileInputDelimited
- tLogRow

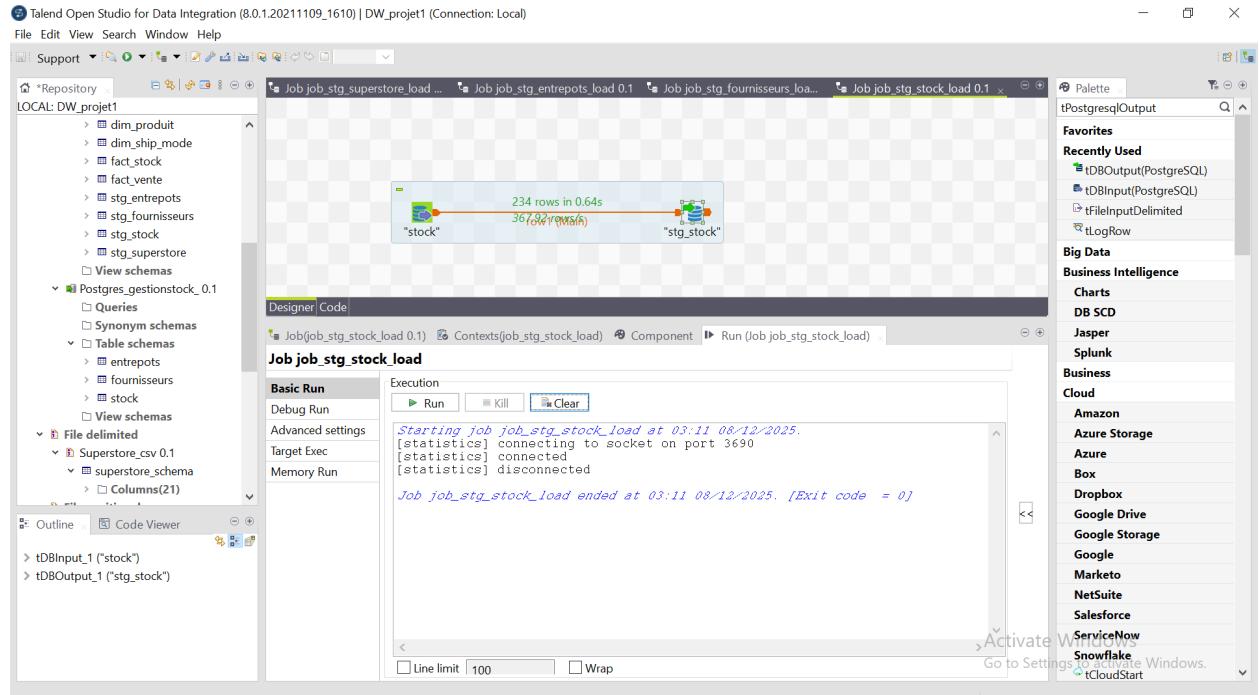
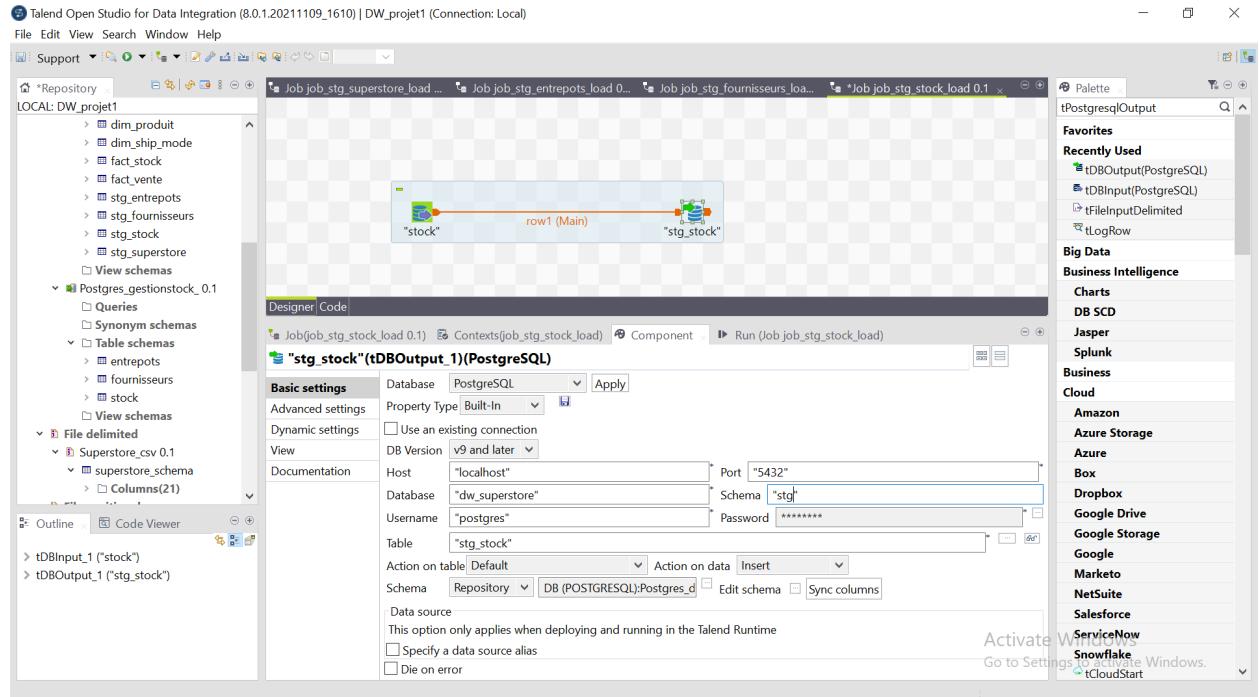
Big Data

Business Intelligence

- Charts
- DB SCD
- Jasper
- Splunk
- Business
- Cloud
- Amazon
- Azure Storage
- Azure
- Box
- Dropbox
- Google Drive
- Google Storage
- Google
- Marketo
- NetSuite
- Salesforce
- ServiceNow
- Snowflake
- tCloudStart

Activate Windows

Go to Settings > Activate Windows.



pgAdmin 4

File Object Tools Edit View Window Help

Object Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock.sql*

Query History

```

1 SELECT COUNT(*) FROM stg.stg_stock;
2 SELECT * FROM stg.stg_stock LIMIT 5;
3

```

Data Output Messages Notifications

	product_id	nom_produit	fournisseur_id	entrepot_id	quantite_stock	date_dernier_reapprovisionnement
1	FUR-BO-10001798	Bush Somerset Collection Bookcase	3	5	45	2024-01-15
2	FUR-BO-10002545	Atlantic Metals Mobile 3-Shelf Bookcases, Custom Colors	3	5	15	2024-02-17
3	FUR-BO-10002613	Atlantic Metals Mobile 4-Shelf Bookcases, Custom Colors	3	5	11	2024-06-03
4	FUR-BO-10004695	O'Sullivan 2-Door Barrister Bookcase in Odessa Pine	3	5	7	2024-04-23
5	FUR-BO-10004834	Riverside Palais Royal Lawyers Bookcase, Royale Cherry Finish	3	5	8	2024-03-22

Activate Windows

Successfully run. Total query runtime: 88 msec. 5 rows affected.

Total rows: 5 Query complete 00:00:00.088 CRLF Ln 3, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock.sql*

Query History

```

1 -- 1) remplir dim_entrepot
2 INSERT INTO dw.dim_entrepot (
3     entrepot_id,
4     nom_entrepot,
5     region
6 )
7     SELECT DISTINCT
8         entrepot_id,
9         nom_entrepot,
10        region
11    FROM stg.stg_entrepots

```

Data Output Messages Notifications

INSERT 0 10

Query returned successfully in 78 msec.

Activate Windows

Query returned successfully in 78 msec.

Total rows: 0 Query complete 00:00:00.078 CRLF Ln 30, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock.sql* Scratch Pad X

```

1 DROP TABLE IF EXISTS dw.fact_stock CASCADE;
2
3 CREATE TABLE dw.fact_stock (
4     stock_key      BIGSERIAL PRIMARY KEY, -- clé technique
5     date_key       INTEGER    NOT NULL,   -- FK > dim_date
6     produit_key    INTEGER    NOT NULL,   -- FK > dim_produit
7     fournisseur_key INTEGER  NOT NULL,   -- FK > dim_fournisseur
8     entrepot_key  INTEGER    NOT NULL,   -- FK > dim_entrepot
9
10    quantite_stock  INTEGER
11
12    CONSTRAINT fk_fact_stock_date
13        FOREIGN KEY (date_key)
14            REFERENCES dw.dim_date (date_key),
15
16    CONSTRAINT fk_fact_stock_produit
17        FOREIGN KEY (produit_key)
18            REFERENCES dw.dim_produit (produit_key),

```

Data Output Messages Notifications

CREATE TABLE

Query returned successfully in 68 msec.

Total rows: Query complete 00:00:00.068

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 10, Col 46

pgAdmin 4

File Object Tools Edit View Window Help

Object Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

```

1 WITH stock_bounds AS (
2     SELECT
3         MIN(date_dernier_reapprovisionnement) AS min_date,
4         MAX(date_dernier_reapprovisionnement) AS max_date
5     FROM stg.stg_stock
6 ),
7 dates AS (
8     SELECT generate_series(min_date, max_date, interval '1 day')::date AS d
9     FROM stock_bounds
10 ),
11 missing_dates AS (
12     SELECT d
13     FROM dates
14     EXCEPT
15     SELECT date
16     FROM stg.stg_stock
17 )

```

Data Output Messages Notifications

INSERT 0 177

Query returned successfully in 92 msec.

Total rows: Query complete 00:00:00.092

Activate Windows
Go to Settings to activate Windows.

CRLF Ln 42, Col 29

pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

```

1 TRUNCATE TABLE dw.fact_stock;
2
3 v INSERT INTO dw.fact_stock (
4   date_key,
5   produit_key,
6   fournisseur_key,
7   entrepot_key,
8   quantite_stock
9 )
10 SELECT
11   d.date_key,
12   p.produit_key,
13   f.fournisseur_key,
14   e.entrepot_key,
15   s.quantite_stock
16 FROM eto_eta_stock_e

```

Data Output Messages Notifications

INSERT 0 173

Query returned successfully in 75 msec.

Total rows: Query complete 00:00:00.075 CRLF Ln 21, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

```

1 SELECT COUNT(*) FROM dw.fact_stock;
2 SELECT * FROM dw.fact_stock LIMIT 5;
3

```

Data Output Messages Notifications

stock_key	[PK] bigint	date_key	integer	produit_key	integer	fournisseur_key	integer	entrepot_key	integer	quantite_stock	integer
1		1	20240115		7		3		5		45
2		2	20240116		769		2		8		500
3		3	20240117		326		1		9		170
4		4	20240118		468		6		7		110
5		5	20240119		495		6		7		95

Showing rows: 1 to 5 Page No: 1 of 1

Total rows: 5 Query complete 00:00:00.164 CRLF Ln 3, Col 1

pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

Servers(1) PostgreSQL 17 Databases(3) dw_superstore gestionstock

Query Query History

```

1 TRUNCATE TABLE dw.fact_stock;
2
3 v INSERT INTO dw.fact_stock (
4   date_key,
5   produit_key,
6   fournisseur_key,
7   entrepot_key,
8   quantite_stock
9 )
10 SELECT
11   d.date_key,
12   p.produit_key,
13   f.fournisseur_key,
14   e.entrepot_key,
15   s.quantite_stock
16 FROM stg.stg_stock s
17 JOIN dw.dim_date d ON d.full_date = s.date_dernier_reapprovisionnement
18 JOIN dw.dim_produit p ON p.product_id = s.product_id
19 JOIN dw.dim_fournisseur f ON f.fournisseur_id = s.fournisseur_id
20 JOIN dw.dim_entrepot e ON e.entrepot_id = s.entrepot_id;

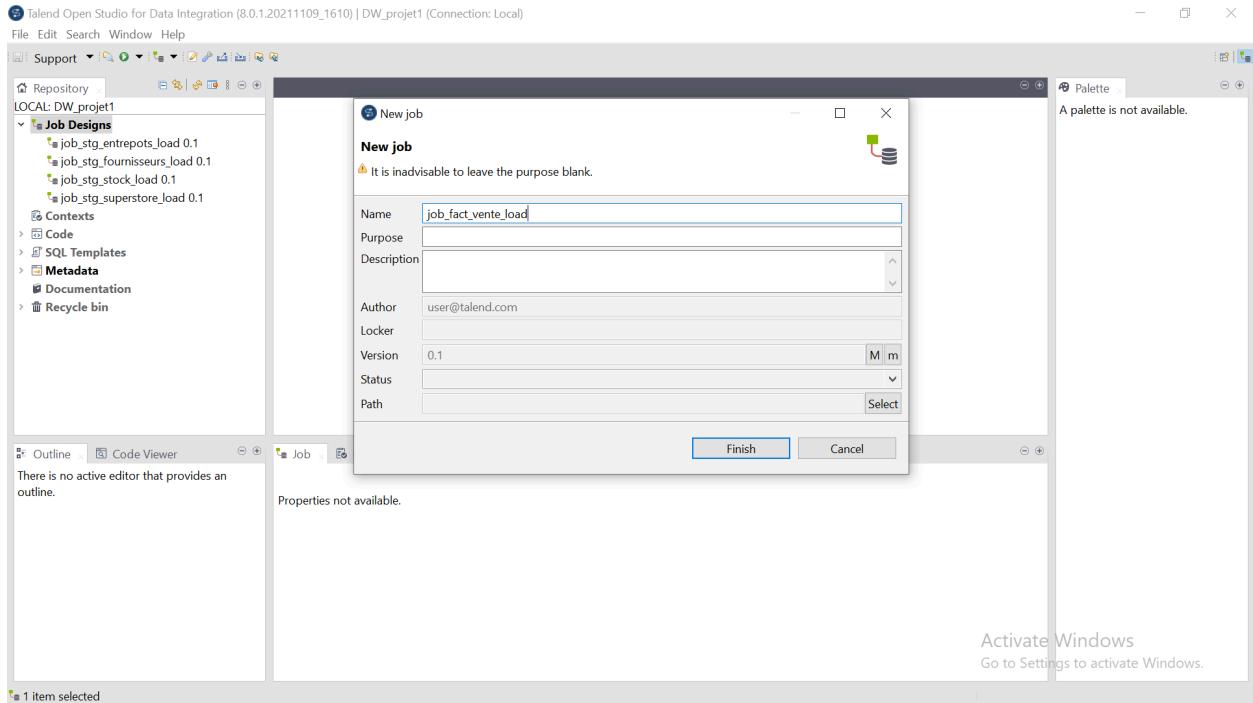
```

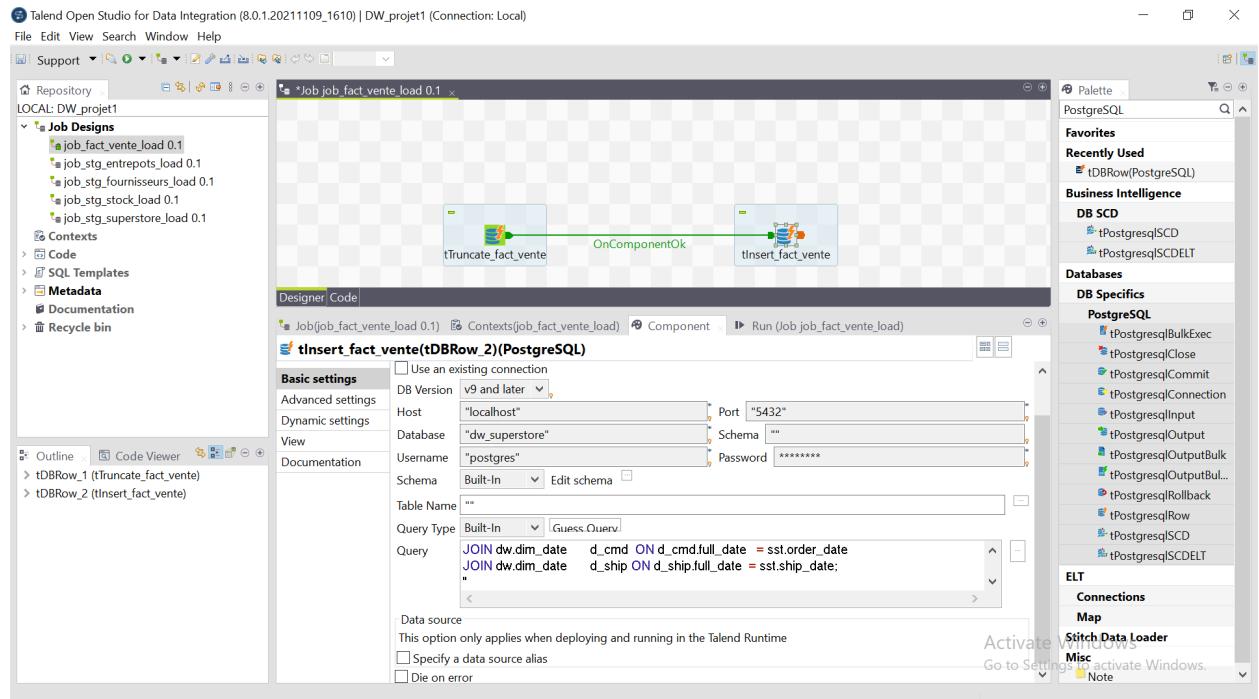
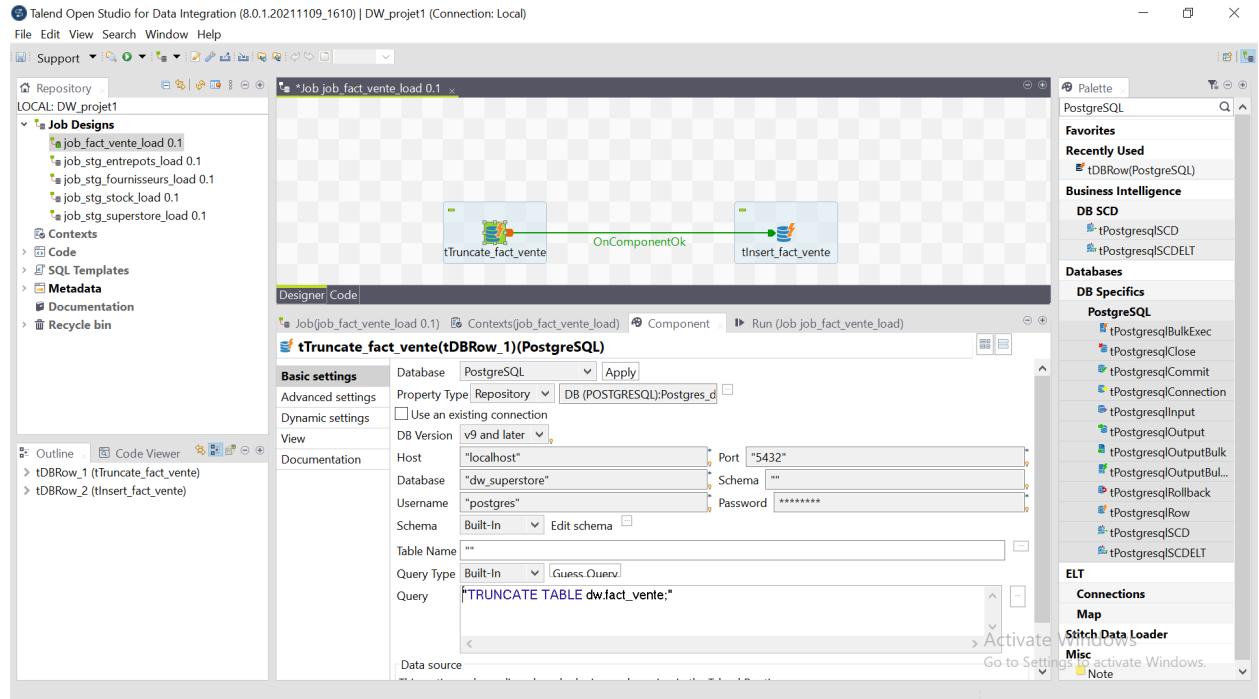
Data Output Messages Notifications

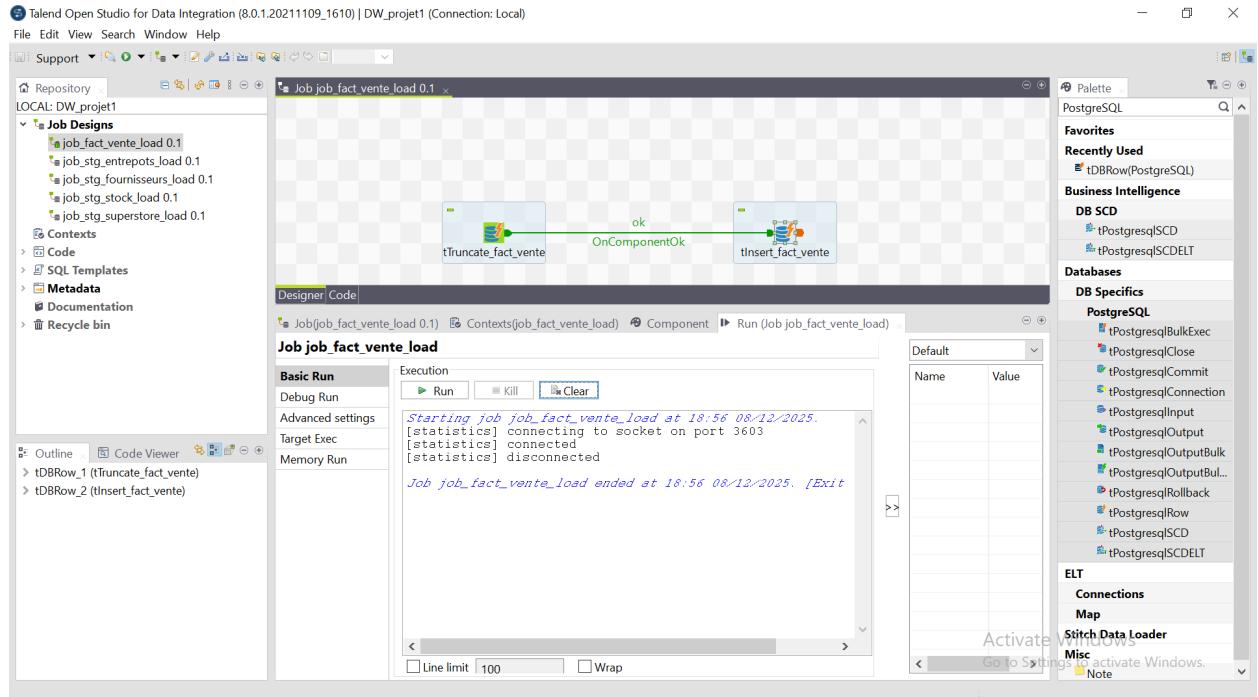
Activate Windows
Go to Settings to activate Windows.

Total rows: 173 Query returned successfully in 86 msec.

CRLF Ln 21, Col 1







pgAdmin 4

File Object Tools Edit View Window Help

Explorer Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post... X

Servers(1)

- PostgreSQL 17
- Databases(3)
 - dw_superstore
 - gestionstock
 - gestionstock_p
- Extensions
- Foreign Data
- Languages
- Publications
- Schemas(1)
 - public
- Aggregates
- Collations
- Domains
- FTS Configurations
- FTS Dictionaries
- FTS Parameters
- FTS Terms
- Foreign Functions
- Materials
- Operational Functions
- Procedures

Query Query History

```

1 SELECT COUNT(*) FROM dw.fact_vente;
2 SELECT * FROM dw.fact_vente LIMIT 10;
3

```

Data Output Messages Notifications

Showing rows: 1 to 10 Page No: 1 of 1

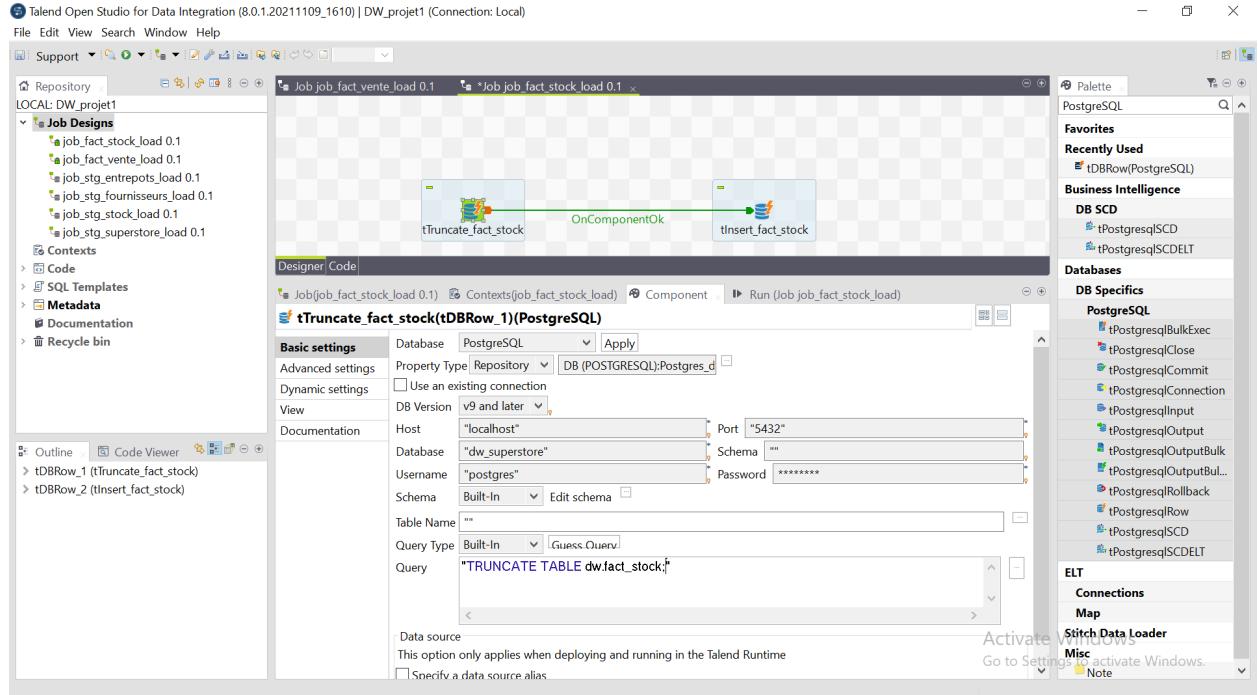
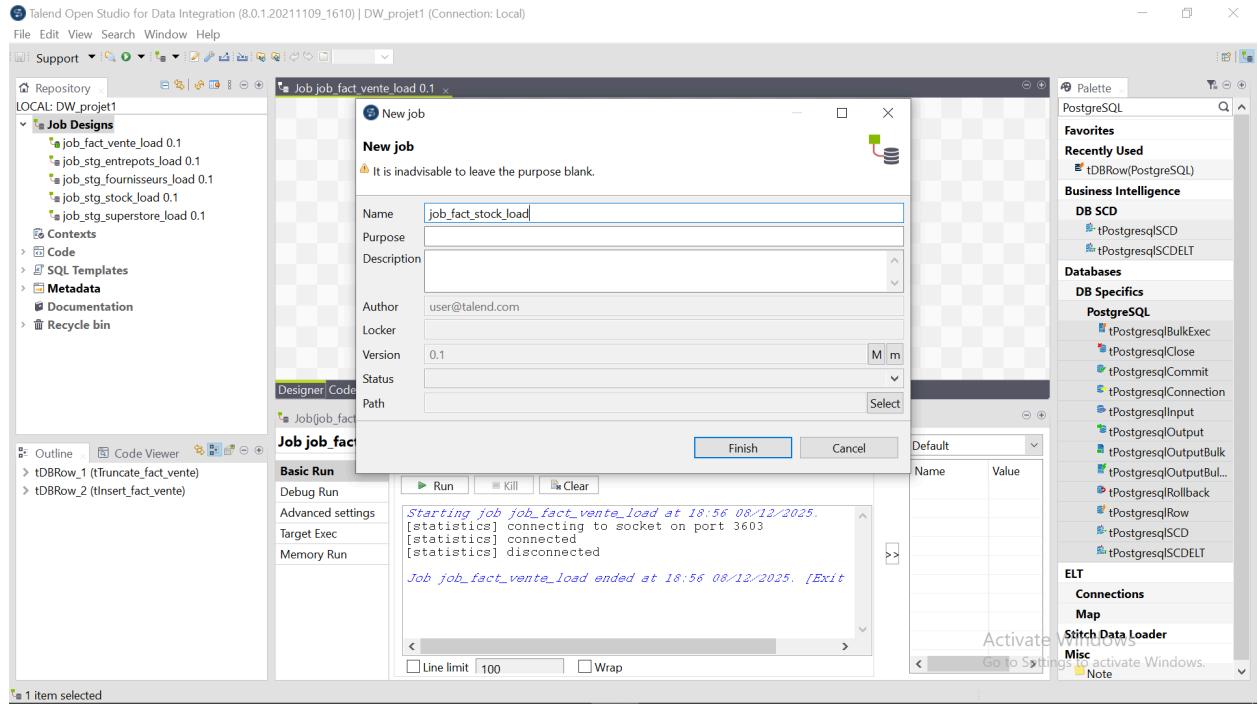
fact_vente_id	[PK] integer	date_commande_key	date_expedition_key	client_key	produit_key	ship_mode_key	order_id	sales	quantity
1	101847	20161108	20161111	744	7	3	CA-2016-152156	261.96	
2	101848	20161108	20161111	743	7	3	CA-2016-152156	261.96	
3	101849	20160612	20160616	1263	691	3	CA-2016-138688	14.62	
4	101850	20160612	20160616	1262	691	3	CA-2016-138688	14.62	
5	101851	20160612	20160616	1261	691	3	CA-2016-138688	14.62	
6	101852	20160612	20160616	1260	691	3	CA-2016-138688	14.62	
7	101853	20151011	20151018	3817	203	4	US-2015-108966	957.58	
8	101854	20151011	20151018	3816	203	4	US-2015-108966	957.58	
9	101855	20151011	20151018	3815	203	4	US-2015-108966	957.58	
10	101856	20151011	20151018	3814	203	4	US-2015-108966	957.58	

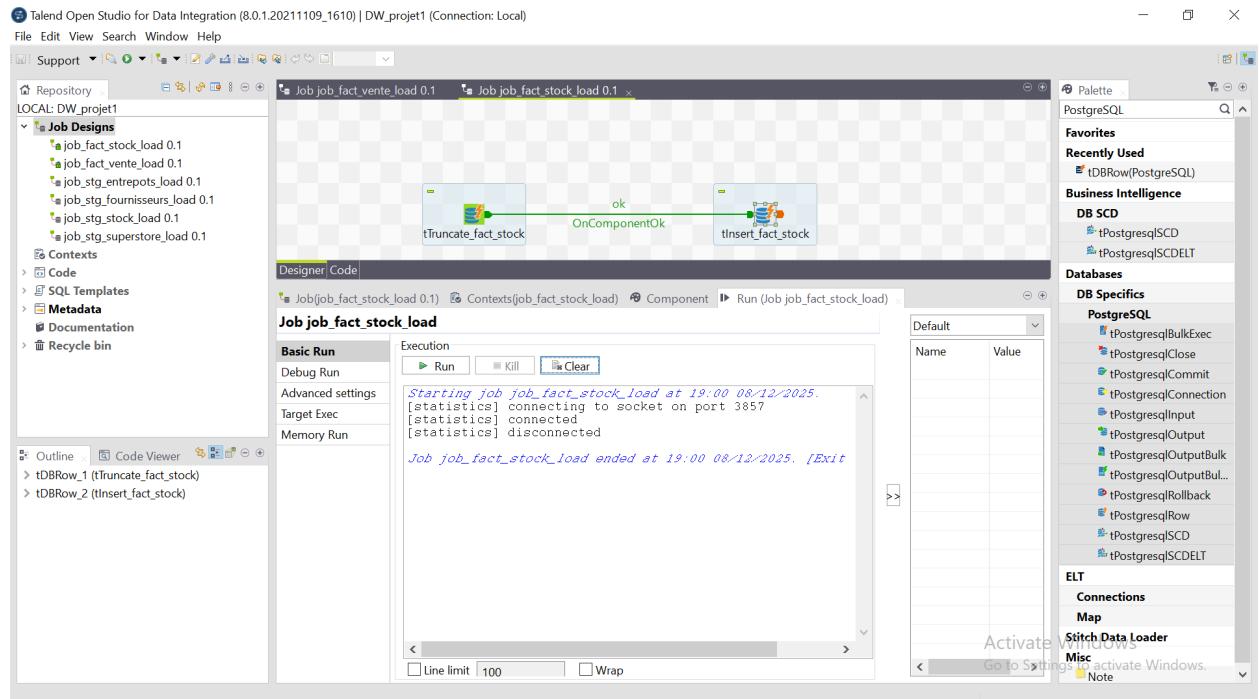
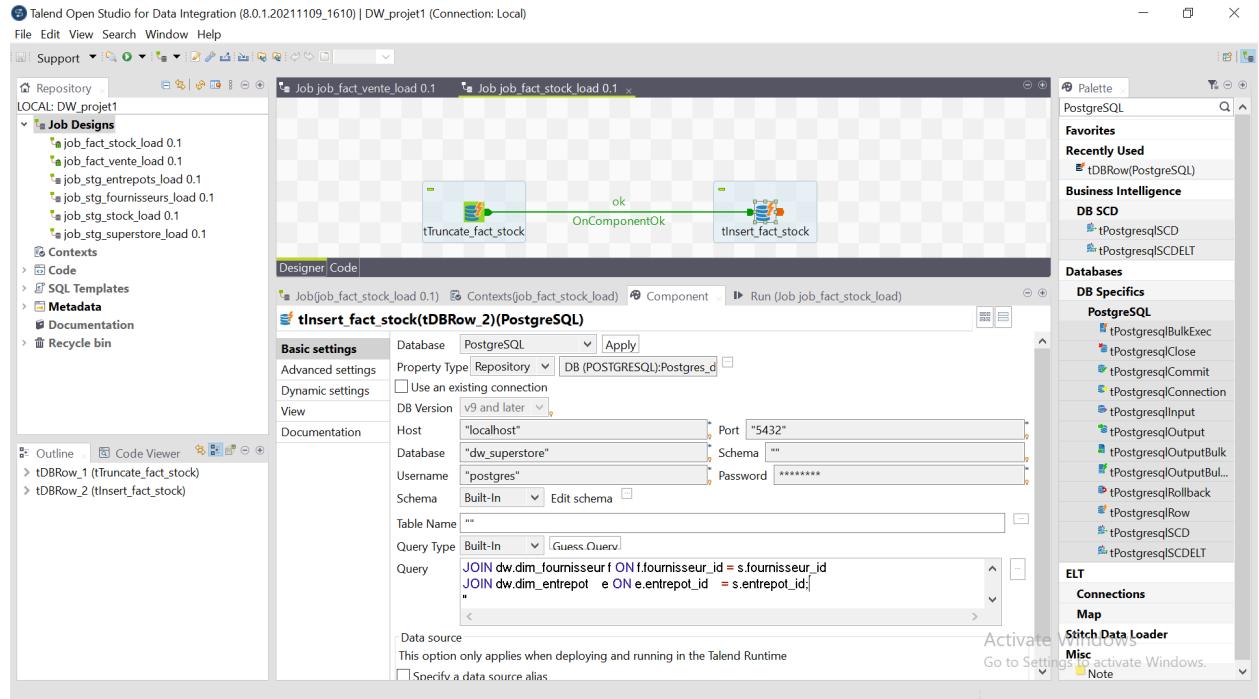
Total rows: 10 Query complete 00:00:00.084

Activate Windows

Go to Settings to activate Windows.

CRLF Ln 3, Col 1





pgAdmin 4

File Object Tools Edit View Window Help

Dashboard dw_superstore/postgres@PostgreSQL 17* gestionstock/post...

Servers(1) PostgreSQL 17 Databases(3) dw_superstore gestionstock

Query History

```
1 SELECT COUNT(*) FROM dw.fact_stock;
2 SELECT * FROM dw.fact_stock LIMIT 10;
3
```

Data Output Messages Notifications

	stock_key [PK] bigint	date_key integer	produit_key integer	fournisseur_key integer	entrepot_key integer	quantite_stock integer
1	347	20240115	7	3	5	45
2	348	20240116	769	2	8	500
3	349	20240117	326	1	9	170
4	350	20240118	468	6	7	110
5	351	20240119	495	6	7	95
6	352	20240120	390	1	9	200
7	353	20240121	461	6	7	130
8	354	20240124	636	2	8	180
9	355	20240125	876	2	8	180
10	356	20240126	1328	10	1	35

Total rows: 10 Query complete 00:00:00.091

Activate Windows

Successfully run. Total query runtime: 91 msec. 10 rows affected.

CRLF Ln 3, Col 1

Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW_projet1 (Connection: Local)

File Edit View Search Window Help

Support Repository Job job_fact_vente_load_0.1 Job job_fact_stock_load_0.1

Job Designs Job job_fact_vente_load_0.1 Job job_fact_stock_load_0.1

New job

It is inadvisable to leave the description blank.

Name: job_parent_dw_superstore

Purpose: Pipeline de jobs

Description:

Author: user@talend.com

Locker:

Version: 0.1

Status:

Path:

Finish Cancel

Basic settings

Advanced settings Property type: Repository DB (POSTGRES):Postgres_d

Dynamic settings Use an existing connection

View DB Version: v9 and later

Documentation Host: "localhost"

Host: "localhost" Port: "5432"

Database: "dw_superstore"

Username: "postgres"

Schema: Built-In

Table Name: ""

Query Type: Built-In Guess Query

Query: "INSERT INTO dw.fact_stock /

Palette PostgreSQL

Favorites Recently Used tDBRow(PostgreSQL)

Business Intelligence DB SCD tPostgresqlSCD tPostgresqlSCDELT

Databases DB Specifics PostgreSQL

ELT

Connections Map

Stitch Data Loader

Misc activate Windows Go to Settings Note

