Guide de réalisation d'un ETL de bout en bout avec Talend Data Integration : mise en place d'un Data Warehouse

– La première partie du mini-projet consiste à mettre en place un système d'extraction des données de suivi des dépenses à l'aide de Talend Studio. Elle permet d'extraire les données depuis un fichier Excel vers plusieurs fichiers plats.

– La deuxième partie du mini-projet, toujours avec Talend Studio, vise à intégrer les données extraites dans les tables de l'Operational Data Store (ODS). Cette étape consiste à charger les données à partir des trois fichiers plats créés précédemment.

– La troisième partie du mini-projet prolonge la précédente et consiste également en une phase d'intégration des données. Elle permet de charger les données depuis les tables de l'ODS vers les tables du Data Warehouse (entrepôt de données), toujours à l'aide de Talend Studio.

Les données sources

Voici les fichiers Excel d'où proviennent les données pour la 1ère partie.
Ces données sont stockées dans un fichier Excel nommée **Saisie_Depense.xlsx**

Liste détaillée des catégories

Catégorie
Dépenses Fixes
Dépenses Variables
Dépenses Exceptionnelles

Liste détaillée des sous-catégories

Sous-catégorie	Catégorie
Abonnement Logiciel	Dépenses Fixes
Abonnement Télécommunications	Dépenses Fixes
Abonnement Transport	Dépenses Fixes
Assurance Maison	Dépenses Fixes
Assurance Voiture	Dépenses Fixes
Autres Abonnements	Dépenses Fixes
Eau	Dépenses Fixes
Énergie (Électricité & Gaz)	Dépenses Fixes
Frais Bancaire	Dépenses Fixes
Impôts et Taxes	Dépenses Fixes
Loyer	Dépenses Fixes
Alimentation	Dépenses Variables
Carburant	Dépenses Variables
Coiffeur	Dépenses Variables
Garderie des Enfants	Dépenses Variables

Liste détaillée des dépenses

Numéro dépense	Date dépense	Description dépense	Catégorie dépense	Sous-catégorie dépense	Montant dépense
D00001	01/01/2022	Achat Essence	Dépenses Variables	Carburant	32,41
D00002	02/01/2022	Alimentation	Dépenses Variables	Alimentation	20,37
D00003	06/01/2022	Abonnement Microsoft	Dépenses Fixes	Abonnement Logiciel	6,48
D00004	09/01/2022	Alimentation	Dépenses Variables	Alimentation	20,82
D00005	11/01/2022	Achat Essence	Dépenses Variables	Carburant	32,41
D00006	12/01/2022	Frais Electricité	Dépenses Fixes	Énergie (Électricité & Gaz)	52,08
D00007	12/01/2022	Frais bancaire	Dépenses Fixes	Frais Bancaire	0,58
D00008	13/01/2022	Soins et Hygiène	Dépenses Variables	Soins et Hygiène	12,93
D00009	13/01/2022	Alimentation	Dépenses Variables	Alimentation	0,97
D00010	16/01/2022	Alimentation	Dépenses Variables	Alimentation	26,19
D00011	16/01/2022	Alimentation	Dépenses Variables	Alimentation	7,44
D00012	16/01/2022	Loisirs	Dépenses Variables	Loisirs	4,54

1ère partie : Système d'extraction de données

Cette première étape du mini projet est réparti en 4 étapes :

- Découverte de Talend
- Création des groupes de contexte
- Création des différents schémas génériques et création des jobs d'extractions des données.
- Création du job principal (Orchestration)

Découverte de Talend data intégration

Travail à réaliser :

– Ouvrir Talend Open Studio.

– On crée un nouveau projet nommé **idepense_reporting** avec l'option Java.

La fenêtre de Talend est composée des vues suivantes :

– Barres d'outils et menus (en haut)

– Repository (en haut à gauche) : Ce référentiel contient tous les éléments techniques du projet

– Design Workspace (au centre) : Cet espace de modélisation permet de concevoir graphiquement les business model et les jobs.

– Palette (en haut à droite) : Cette palette graphique permet d'accéder aux différents composants

– Différentes vues (en bas au centre) :

- Job : infos sur le job sélectionné
- Component : configuration du composant sélectionné
- Run job : exécution des jobs
- Problems : erreurs

– Outline et Code Viewer (en bas à gauche) : Ces fenêtres fournissent un aperçu du code et du schéma du job ou du business model.

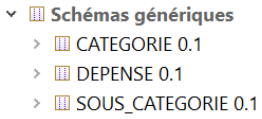
Création des groupes de contexte :

- On crée un nouveau groupe de contexte nommé **config_Depense_File**
- Initialisation des variables :
  - **depenseFileRepo :** récupère le dossier qui contient le fichier source « INPUT/ »
  - **depenseFileName** : récupère le nom de fichier source « SAISIE_DEPENSE.xlsx »
  - **projetFolder :** récupère le nom du dossier du projet « c:/bi/depense/ »
  - **encodage :** récupère l'encodage des fichiers « UTF8 »
  - **fieldSeparator :** récupère les séparateurs de champs « | »
  - **dataFolder :** récupère les données du dossier « DATA/ »

Création des différents schémas génériques : Fichiers plats

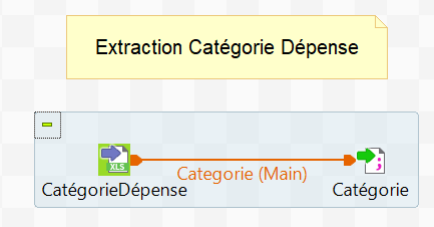
Dans la métadonnée clic droit sur schémas générique puis créer un schéma générique

- CATEGORIE
- DEPENSE
- SOUS CATEGORIE

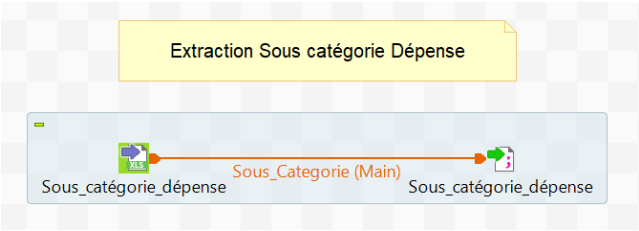


Création des jobs d'extraction des données (Lecture du fichier Excel et extraction des données vers les fichiers plats)

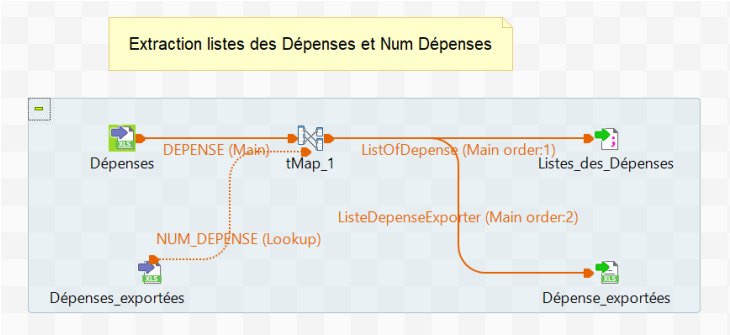
Un job nommé jExtractDepense : extraction catégorie dépense :



Un job nommé jExtractSousCategorieDepense : Extraction Sous-catégorie dépense



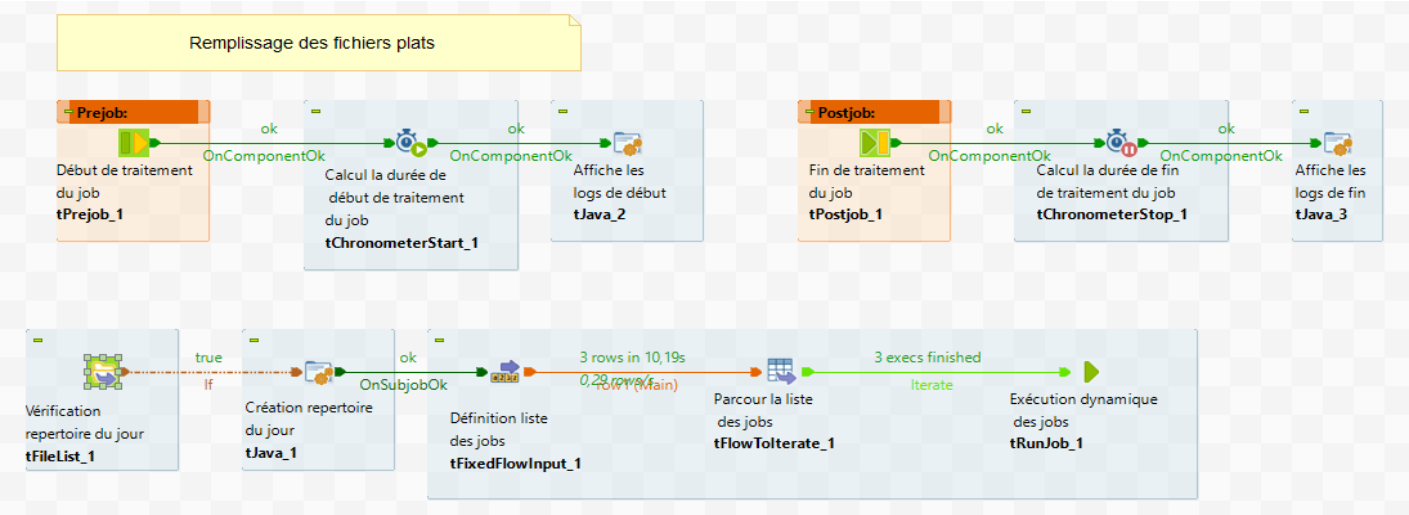
Un job nommé jExtractCategorieDepense : extraction des dépenses :



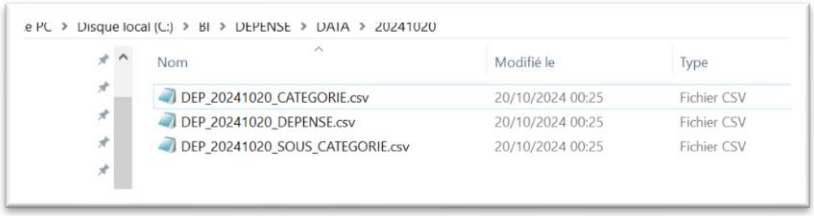
Orchestration des jobs :

Un dernier job, nommé *jExtractData*, est nécessaire pour orchestrer l'ensemble des jobs d'extraction créés précédemment. Lors de son exécution, ce job appelle successivement les jobs *jExtractDepense*, *jExtractSousCategorieDepense* et *jExtractCategorieDepense*. Ci-dessous, un aperçu de l'ordonnancement de ces jobs d'extraction.

Création du job principal(**jExtractData**) : remplissage des fichiers plats.



Ci-dessous les 3 fichiers horodatés générés :



Conclusion

Le système d'extraction mis en place permet de récupérer les données depuis un fichier Excel, puis de générer trois fichiers plats à partir de ces données. La génération des fichiers se fait dans un répertoire horodaté, ce qui nous a amenés à définir des répertoires horodatés pour organiser les extractions.

2ème partie : Chargement des tables ODS (Operational Data Store) –Suivi des dépenses

La deuxième partie du mini-projet, toujours avec Talend Studio, vise à intégrer les données extraites dans les tables de l'Operational Data Store (ODS). Cette étape consiste à charger les données à partir des trois fichiers plats créés précédemment. Cette partie comprend 9 étapes :

- Création de la base de données via PostgreSQL
- Création des différents schémas (ODS, PARAM_log)
- Création des tables ODS ET table des CONTEXTES
- Mise en place des différents processus d'intégration de données talend
- Création des groupes de contexte
- Création des métadonnées : Connexion à la base de données, Configuration du chargement implicite des variables de contexte
- Création des jobs ODS (Transformation et Chargement de la donnée)
- Création du job principal (Orchestration)

Création de la base de données "IDEPENSE_INGESTION_DB" qui permet d'héberger les données

```
-- Création de la base de données "IDEPENSE_INGESTION_DB"
DROP DATABASE IF EXISTS "IDEPENSE_INGESTION_DB";
CREATE DATABASE "IDEPENSE_INGESTION_DB";
```

Création du schéma DEPENSE_ODS :

```
DROP SCHEMA IF EXISTS "DEPENSE_ODS";
CREATE SCHEMA "DEPENSE_ODS";
```

Création du schéma PARAMS_LOG :

```
DROP SCHEMA IF EXISTS "PARAMS_LOG";
CREATE SCHEMA "PARAMS_LOG";
```

Création des tables ODS et contexte :

Table ODS_CATEGORIE:

```
DROP TABLE IF EXISTS "ODS_CATEGORIE" ;
CREATE TABLE "ODS_CATEGORIE"
(
"LB_CATEGORIE"    VARCHAR(100) NOT NULL,
"LB_NOM_FICHIER"  VARCHAR(100) NOT NULL,
"DT_INSERTION"    TIMESTAMP    NOT NULL,
"LB_JOB_NAME"     VARCHAR(100) NOT NULL,
CONSTRAINT "ODS_CATEGORIE_PKEY" PRIMARY KEY ("LB_CATEGORIE")
);
```

Table ODS_SOUS_CATEGORIE :

```
DROP TABLE IF EXISTS "ODS_SOUS_CATEGORIE" ;
CREATE TABLE "ODS_SOUS_CATEGORIE"
(
"LB_SOUS_CATEGORIE" VARCHAR(100) NOT NULL,
"LB_CATEGORIE"      VARCHAR(50)  NOT NULL,
"LB_NOM_FICHIER"    VARCHAR(100) NOT NULL,
"DT_INSERTION"      TIMESTAMP    NOT NULL,
"LB_JOB_NAME"       VARCHAR(100) NOT NULL,
CONSTRAINT "ODS_SOUS_CATEGORIE_PKEY" PRIMARY KEY ("LB_SOUS_CATEGORIE")
);
```

Table ODS_SOUS_DESCRIPTION :

```
DROP TABLE IF EXISTS "ODS_DESCRIPTION" ;
CREATE TABLE "ODS_DESCRIPTION"
(
"LB_DESCRIPTION_DEPENSE"    VARCHAR(255) NOT NULL,
"LB_NOM_FICHIER"            VARCHAR(100) NOT NULL,
"DT_INSERTION"              TIMESTAMP    NOT NULL,
"LB_JOB_NAME"               VARCHAR(100) NOT NULL,
CONSTRAINT "ODS_DESCRIPTION_PKEY" PRIMARY KEY ("LB_DESCRIPTION_DEPENSE")
);
```

Table ODS_DEPENSE :

```
DROP TABLE IF EXISTS "ODS_DEPENSE" ;
CREATE TABLE "ODS_DEPENSE"
(
"NUM_DEPENSE"               VARCHAR(50)   NOT NULL,
"DT_DEPENSE"                TIMESTAMP     NOT NULL,
"LB_DESCRIPTION_DEPENSE"    VARCHAR(255)  NOT NULL,
"LB_SOUS_CATEGORIE"         VARCHAR(50)   NOT NULL,
"MT_DEPENSE"                NUMERIC       NOT NULL,
"LB_NOM_FICHIER"            VARCHAR(100)  NOT NULL,
"DT_INSERTION"              TIMESTAMP     NOT NULL,
"LB_JOB_NAME"               VARCHAR(100)  NOT NULL,
CONSTRAINT "ODS_DEPENSE_PKEY" PRIMARY KEY ("NUM_DEPENSE")
);
```

Table ODS_REJET :

```
DROP TABLE IF EXISTS "ODS_REJET" ;
CREATE TABLE "ODS_REJET"
(
"LB_CHEMIN_FICHIER" VARCHAR(255) NOT NULL,
"LB_NOM_FICHIER"    VARCHAR(100) NOT NULL,
"NUM_LIGNE_REJET"   INTEGER      NOT NULL,
"LB_LIGNE_REJET"    VARCHAR(500) NOT NULL,
"LB_MESSAGE_REJET"  VARCHAR(100) NOT NULL,
"LB_NOM_FLUX"       VARCHAR(100) NOT NULL,
"DT_REJET"          TIMESTAMP    NOT NULL,
"LB_JOB_NAME"       VARCHAR(100) NOT NULL,
CONSTRAINT "ODS_REJET_PKEY" PRIMARY KEY
("LB_CHEMIN_FICHIER", "LB_NOM_FICHIER", "NUM_LIGNE_REJET", "LB_LIGNE_REJET")
);
```

Table CONTEXTE :

```
DROP TABLE IF EXISTS "CONTEXTE";
CREATE TABLE "CONTEXTE"
(
    key   VARCHAR(100) NOT NULL,
    value VARCHAR(255) ,
    CONSTRAINT "CONTEXTE_PKEY" PRIMARY KEY (key)
);
```

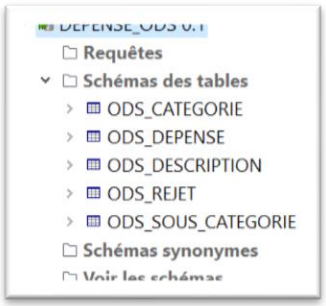
Mise en place des différents processus d'intégration de données via Talend :

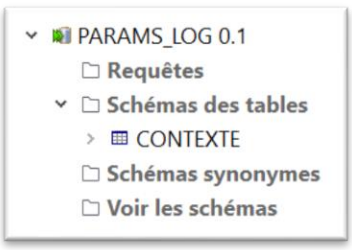
- Création des groupes de contexte :
  - On crée un nouveau groupe de contexte nommé **Connexion_ODS** sur Talend
  - Initialisation des variables :
    - **serverName :** récupère le nom de serveur
    - **database**: récupère le nom de la BD « IDEPENSE_INGESTION_DB »
    - **port :** récupère le port
    - **password :** récupère le mot de passe
    - **utilisateur :** récupère le nom utilisateur
    - **additionalParam :** champ vide
    - **schema_ods :** récupère le schéma des « DEPENSE_ODS »

Reproduire les mêmes informations pour le groupe de contexte **Connexion_Params**

- **serverName :** récupère le nom de serveur
- **database**: récupère le nom de la BD « IDEPENSE_INGESTION_DB »
- **Port :** récupère le port
- **password :** récupère le mot de passe
- **utilisateur :** récupère le nom utilisateur
- **additionalParam :** champ vide
- **schema_param:** récupère le schéma « PARAMS_LOG »

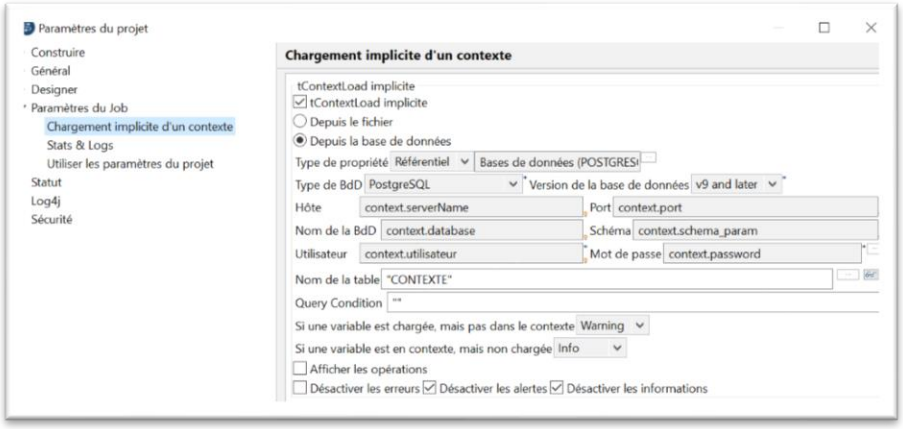
Création des métadonnées : Connexion à la base de données et récupérer le schéma



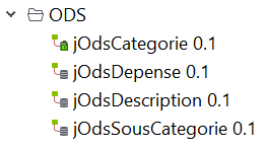


Configuration du chargement implicite des variables de contexte :

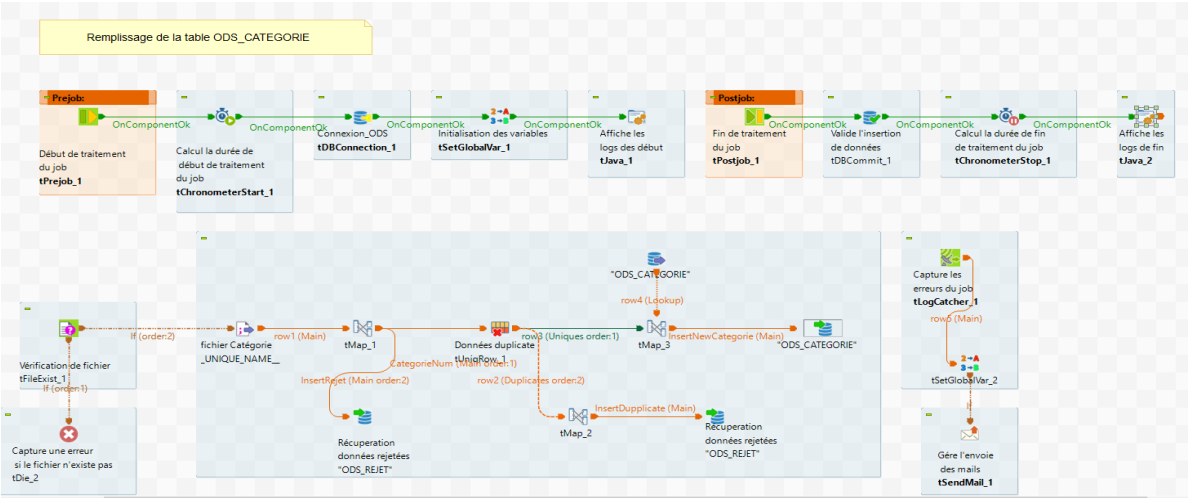
- Ouvrir Talend-Fichier-modifier les propriétés du job-paramètre du job puis chargement implicite d'un contexte



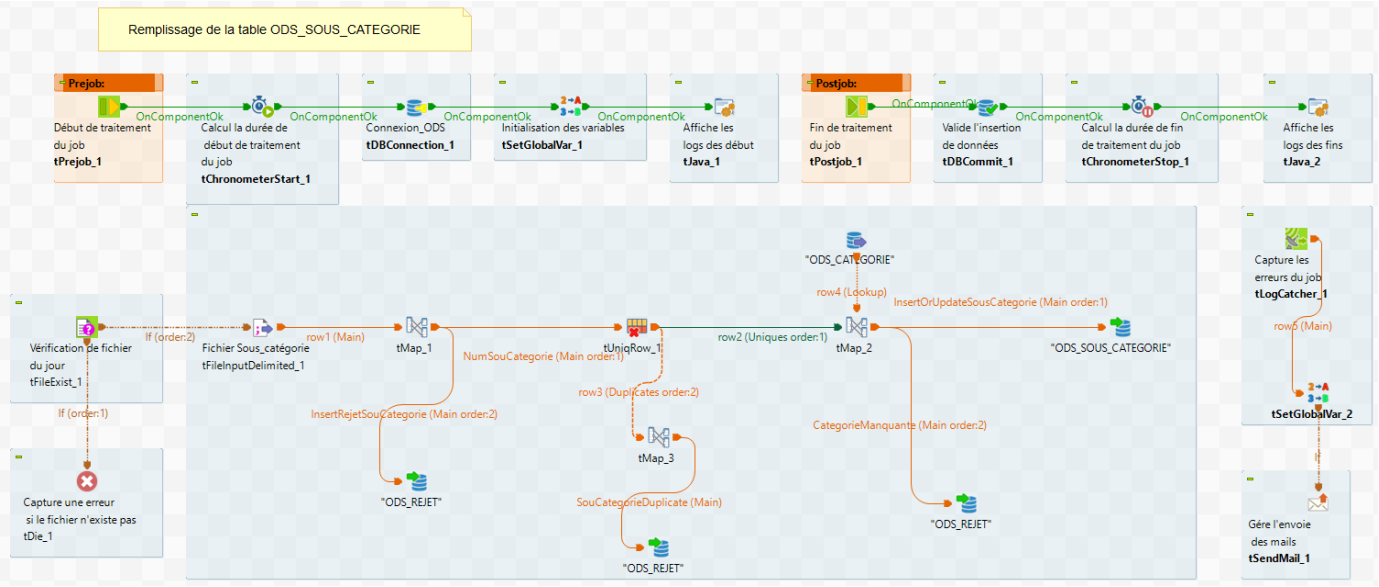
Création des jobs ODS (Transformation et Chargement de la donnée) :



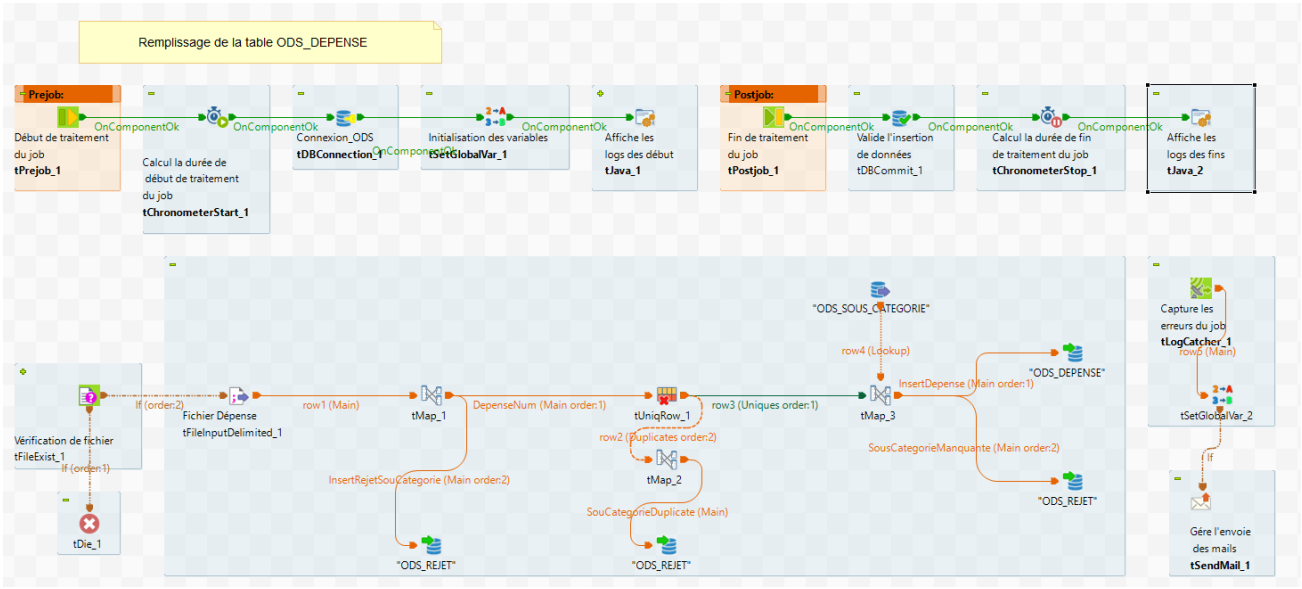
Le job nommé jOds_Categorie: remplissage de la table ODS_CATEGORIE



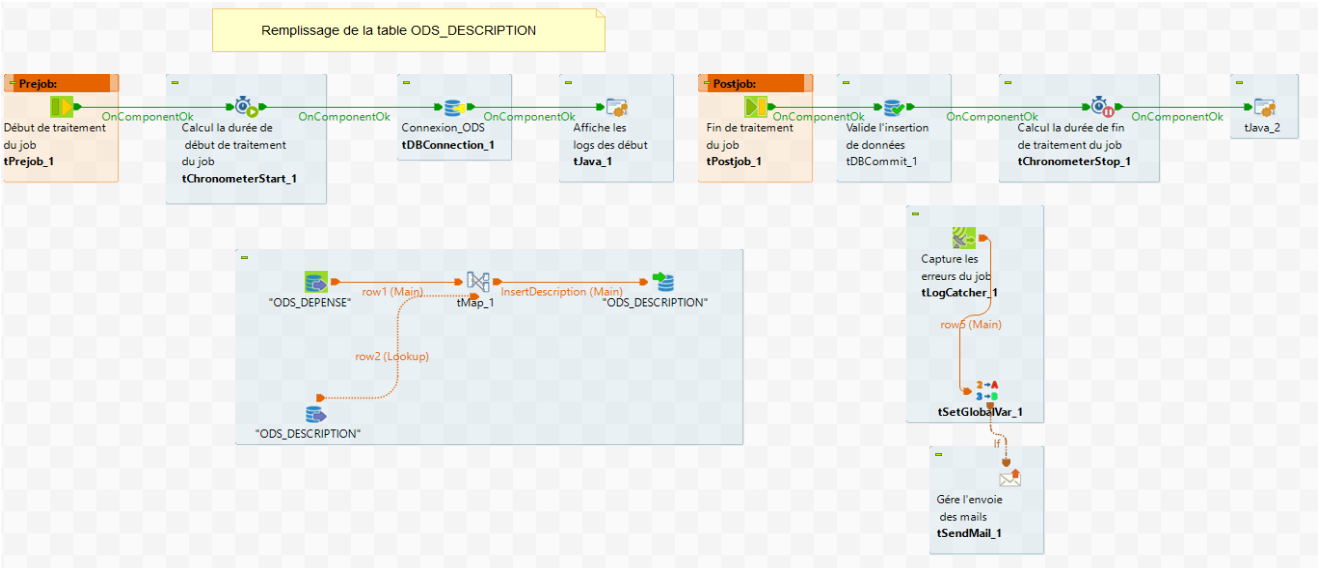
Le job nommé jOds_Sous_Categorie : remplissage de la table ODS_SOUS_CATEGORIE



Le job nommé jOds_Depense : remplissage de la table ODS_DEPENSE

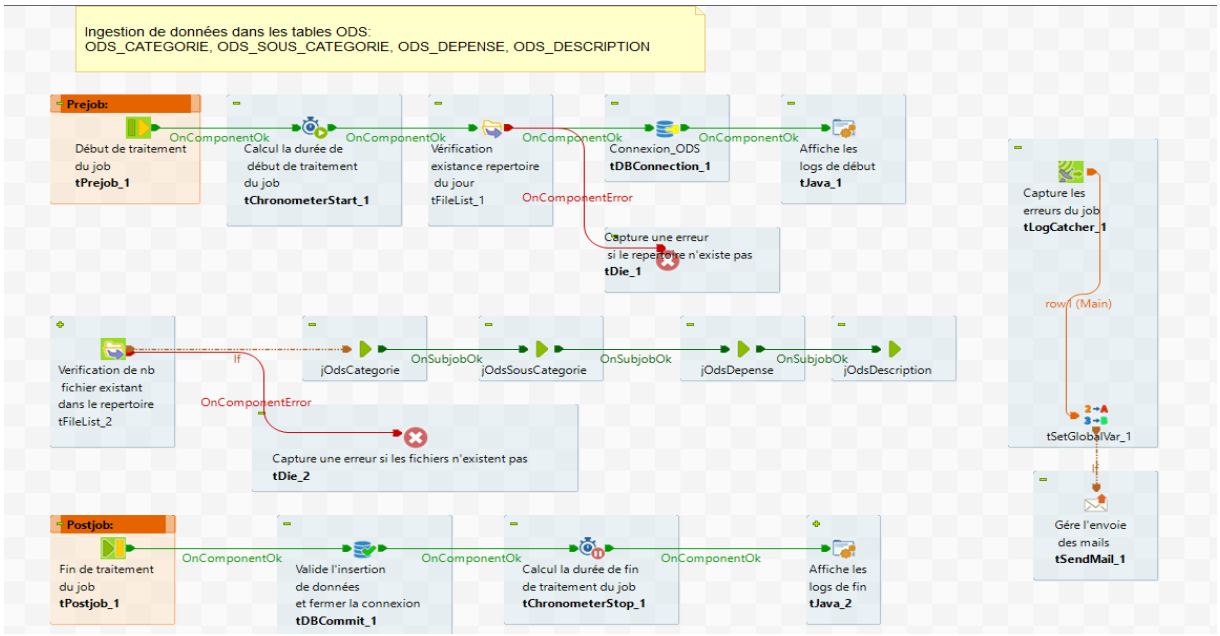


Le job nommé jOds_Description : remplissage de la table ODS_DESCRIPTION



Création du job principal (Orchestration ODS)

Le job nommé jChargeODS :

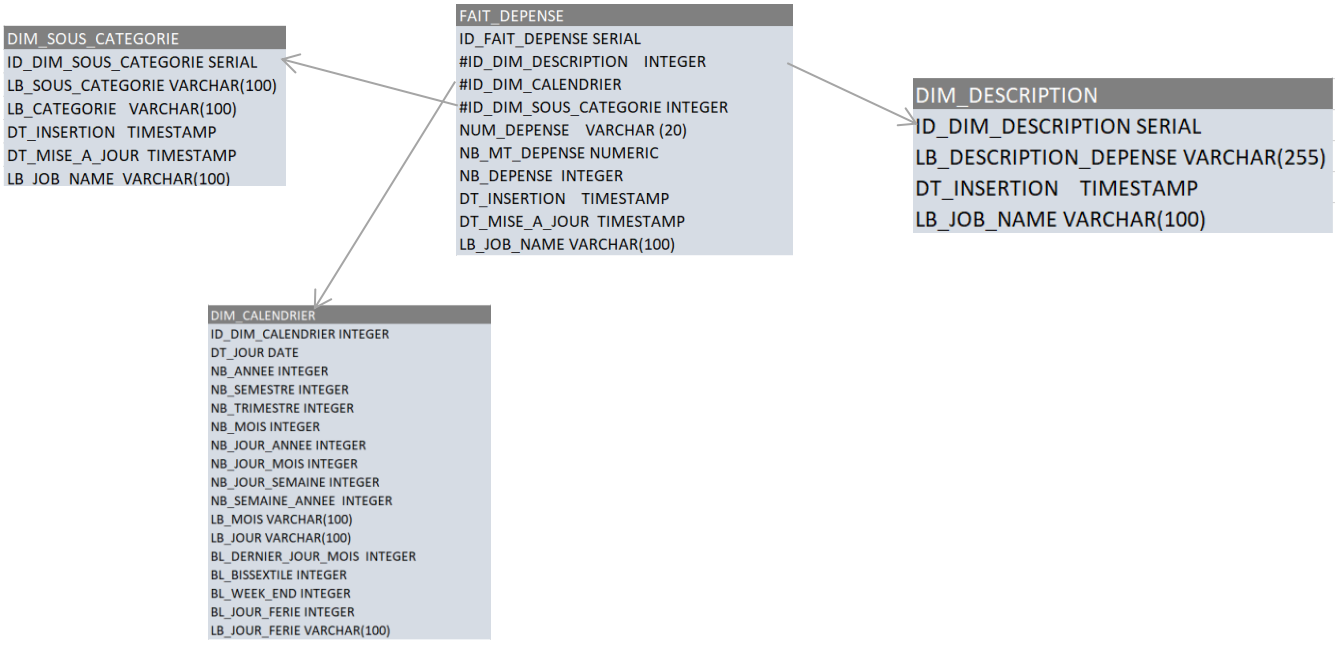


3ème partie : Chargement de Data Warehouse (Entrepôt de Données) –Suivi des dépenses.

La troisième partie du mini-projet prolonge la précédente et consiste également en une phase d'intégration des données. Elle permet de charger les données depuis les tables de l'ODS vers les tables du Data Warehouse (entrepôt de données), toujours à l'aide de Talend Studio.

Schéma de l'entrepôt de données (DWH)

Voici le schéma en étoile de l'entrepôt de données :



Une fois le schéma validé, il faut créer l'entrepôt sous PostgreSQL.

Cette partie est répartie en 7 étapes :

- Création des différents schémas (DEPENSE_DWH)
- Création des tables des dimensions
- Mise en place des différents processus d'intégration de données via Talend Data intégration
- Création des groupes de contexte
- Création des métadonnées : Connexion à la base de données, et récupération des schémas
- Création des jobs DWH (Transformation et Chargement de la donnée)
- Création du job principal (Orchestration).

Création des différents schémas (DEPENSE_DWH)

```
DROP SCHEMA IF EXISTS "DEPENSE_DWH" CASCADE;
CREATE SCHEMA "DEPENSE_DWH";
```

Création des tables des dimensions :

Table de dimension : DIM_SOUS_CATEGORIE

```sql
-- DROP TABLE IF EXISTS "DIM_SOUS_CATEGORIE" ;
CREATE TABLE "DIM_SOUS_CATEGORIE"
(
    "ID_DIM_SOUS_CATEGORIE" SERIAL       NOT NULL,
    "LB_SOUS_CATEGORIE"     VARCHAR(100) NOT NULL,
    "LB_CATEGORIE"          VARCHAR(100) NOT NULL,
    "DT_INSERTION"          TIMESTAMP    NOT NULL,
    "LB_JOB_NAME"           VARCHAR(100) NOT NULL,
    CONSTRAINT "DIM_SOUS_CATEGORIE_PKEY" PRIMARY KEY ("ID_DIM_SOUS_CATEGORIE")
);
```

Table de dimension : DIM_DESCRIPTION

```sql
-- DROP TABLE IF EXISTS "DIM_DESCRIPTION" ;
CREATE TABLE "DIM_DESCRIPTION"
(
    "ID_DIM_DESCRIPTION"        SERIAL       NOT NULL,
    "LB_DESCRIPTION_DEPENSE"    VARCHAR(255) NOT NULL,
    "DT_INSERTION"              TIMESTAMP    NOT NULL,
    "LB_JOB_NAME"               VARCHAR(100) NOT NULL,
    CONSTRAINT "DIM_DESCRIPTION_PKEY" PRIMARY KEY ("ID_DIM_DESCRIPTION")
);
```

Table de dimension : DIM_CALENDRIER

```sql
-- DROP TABLE IF EXISTS "DIM_CALENDRIER";
CREATE TABLE "DIM_CALENDRIER"
(
    "ID_DIM_CALENDRIER"     INTEGER NOT NULL,
    "DT_JOUR"               DATE NOT NULL,
    "NB_ANNEE"              INTEGER NOT NULL,
    "NB_SEMESTRE"           INTEGER NOT NULL,
    "NB_TRIMESTRE"          INTEGER NOT NULL,
    "NB_MOIS"               INTEGER NOT NULL,
    "NB_JOUR_ANNEE"         INTEGER NOT NULL,
    "NB_JOUR_MOIS"          INTEGER NOT NULL,
    "NB_JOUR_SEMAINE"       INTEGER NOT NULL,
    "NB_SEMAINE_ANNEE"      INTEGER NOT NULL,
    "LB_MOIS"               VARCHAR(100) NOT NULL,
    "LB_JOUR"               VARCHAR(100) NOT NULL,
    "BL_DERNIER_JOUR_MOIS"  INTEGER NOT NULL,
    "BL_BISSEXTILE"         INTEGER NOT NULL,
    "BL_WEEK_END"           INTEGER NOT NULL,
    "BL_JOUR_FERIE"         INTEGER NOT NULL,
    "LB_JOUR_FERIE"         VARCHAR(100),
    CONSTRAINT "DIM_CALENDRIER_PKEY" PRIMARY KEY ("ID_DIM_CALENDRIER")
);
```

Table de faits : FAI_DEPENSE

```sql
-- DROP TABLE IF EXISTS "FAIT_DEPENSE" ;
CREATE TABLE "FAIT_DEPENSE"
(
    "ID_FAIT_DEPENSE"       SERIAL    NOT NULL,
    "ID_DIM_CALENDRIER"     INTEGER   NOT NULL,
    "ID_DIM_DESCRIPTION"    INTEGER   NOT NULL,
    "ID_DIM_SOUS_CATEGORIE" INTEGER   NOT NULL,
    "NUM_DEPENSE"           VARCHAR   NOT NULL,
    "NB_MT_DEPENSE"         NUMERIC   NOT NULL,
    "NB_DEPENSE"            INTEGER   NOT NULL,
    "DT_INSERTION"          TIMESTAMP NOT NULL,
    "DT_MISE_A_JOUR"        TIMESTAMP ,
    "LB_JOB_NAME"           VARCHAR(100) NOT NULL,
    CONSTRAINT "FAIT_DEPENSE_PKEY" PRIMARY KEY ("ID_FAIT_DEPENSE"),
    CONSTRAINT "FAIT_DEPENSE_SOUS_CATEGORIE_FKEY" FOREIGN KEY ("ID_DIM_SOUS_CATEGORIE")
            REFERENCES "DIM_SOUS_CATEGORIE" ("ID_DIM_SOUS_CATEGORIE"),
    CONSTRAINT "FAIT_DEPENSE_CALENDRIER_FKEY" FOREIGN KEY ("ID_DIM_CALENDRIER")
            REFERENCES "DIM_CALENDRIER" ("ID_DIM_CALENDRIER"),
    CONSTRAINT "FAIT_DEPENSE_DESCRIPTION_FKEY" FOREIGN KEY ("ID_DIM_DESCRIPTION")
            REFERENCES "DIM_DESCRIPTION" ("ID_DIM_DESCRIPTION")
);
```

Les requêtes ont été exécutées avec succès sous PostgreSQL pour créer les tables.
Vérification du résultat de création des tables sous PostgreSQL en lançant la requête ci-dessous :

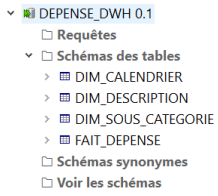
```sql
SELECT table_name
FROM information_schema.tables
WHERE table_schema = 'DEPENSE_DWH';
```

	table_name name
1	DIM_SOUS_CATEGORIE
2	FAIT_DEPENSE
3	DIM_CALENDRIER
4	DIM_DESCRIPTION

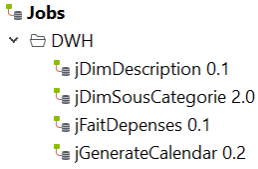
Mise en place des différents processus d'intégration de données via Talend studio

- Création des groupes de contexte :
  - On crée un nouveau groupe de contexte nommé **Connexion_DWH** via Talend
  - Initialisation des variables :
    - **serverName :** récupère le nom de serveur
    - **database**: récupère le nom de la base de données
    - **port :** récupère le port
    - **password :** récupère le mot de passe
    - **utilisateur :** récupère le nom utilisateur
    - **additionalParam :** champ vide
    - **schema_dwh:** récupère le schéma des « Depense_dwh »

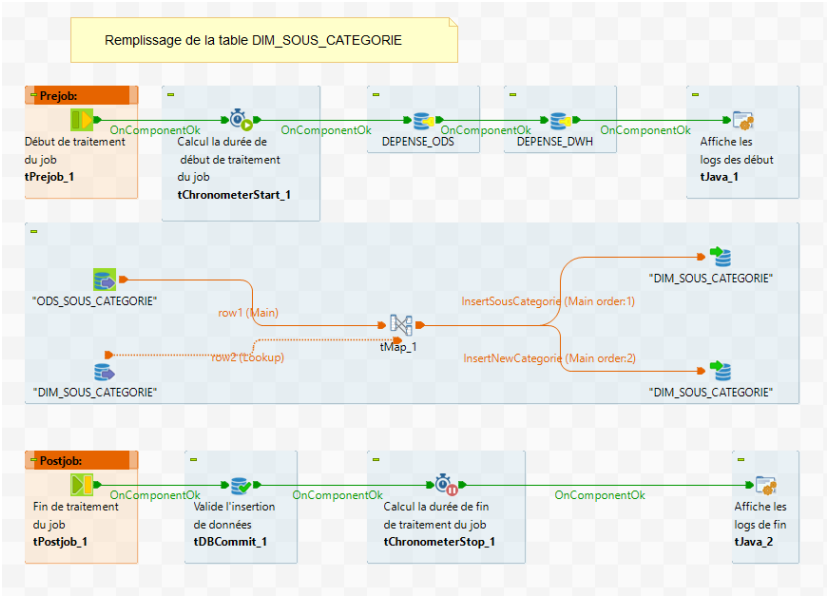
Création des métadonnées : Connexion à la base de données, et récupération des schémas



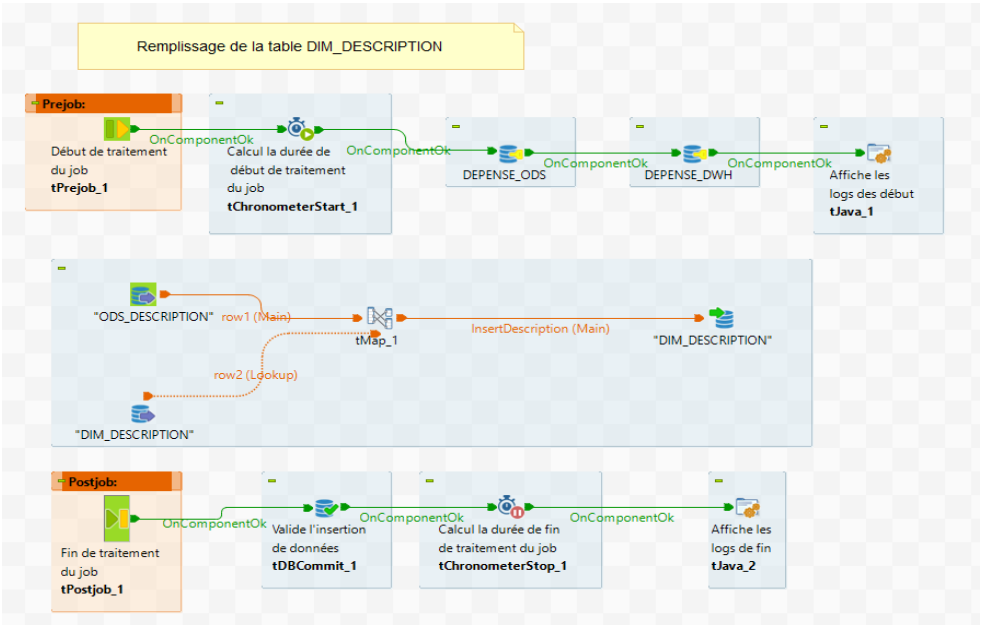
Création des jobs DWH (Transformation et Chargement de la donnée) :



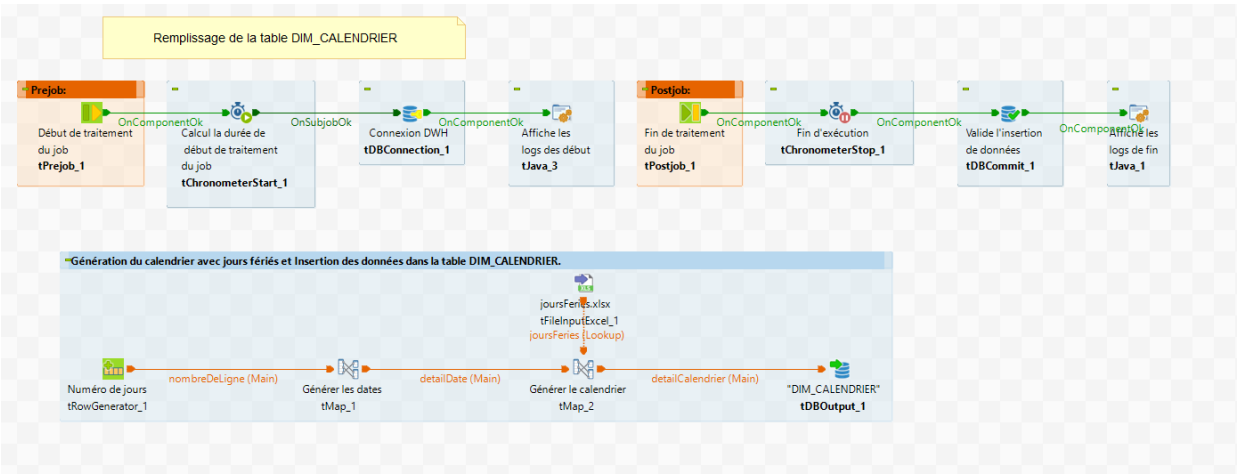
Le job nommé jDim_Sous_Categorie : Remplissage de la table DIM_SOUS_CATEGORIE



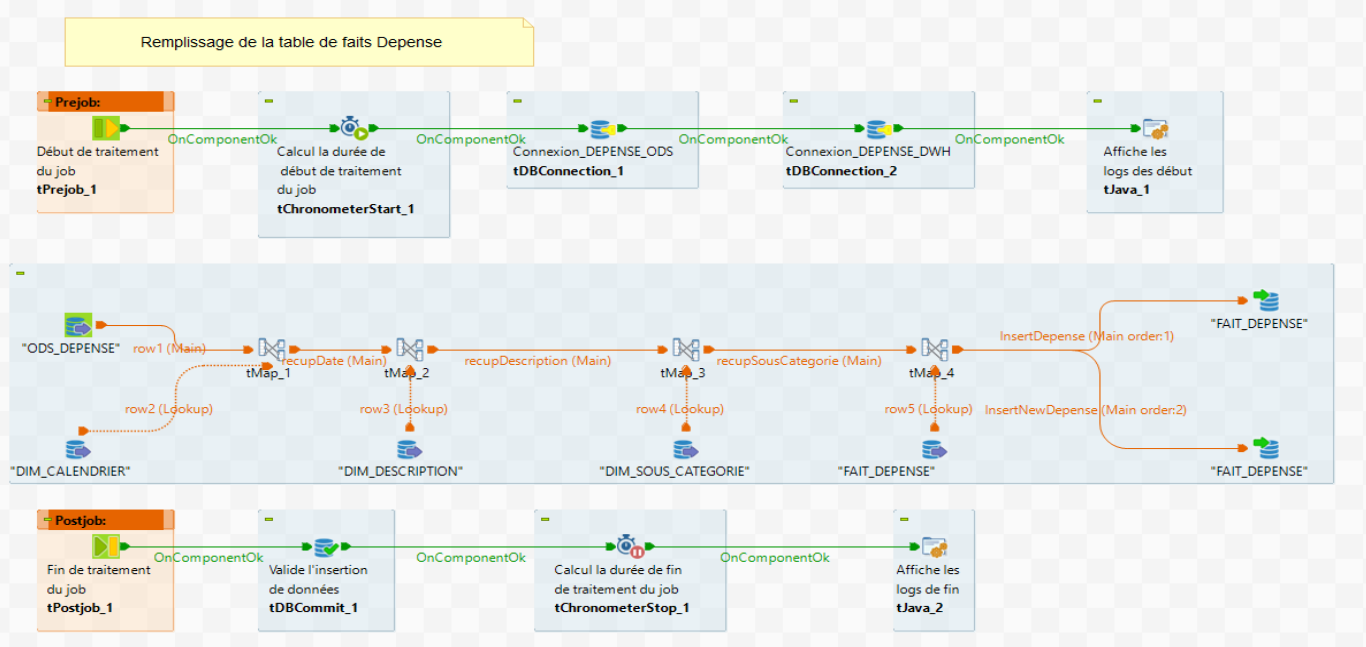
Le job nommé jDim_Description : Remplissage de la table DIM_DESCRIPTION



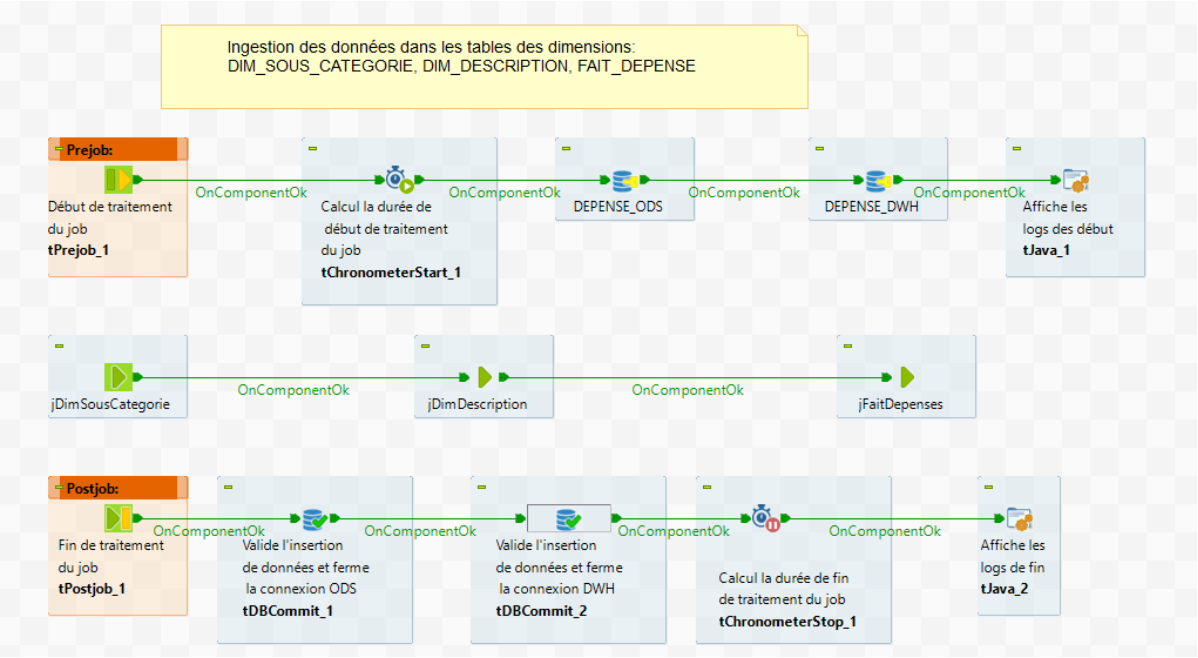
Le job nommé jGenerate_Calendar: Remplissage de la table DIM_CALENDRIER



Le job nommé jFait_Depenses: Remplissage de la table FAIT_DEPENSE



Création du job principal (Orchestration DWH) :



Orchestration (Alimentation BDD) :

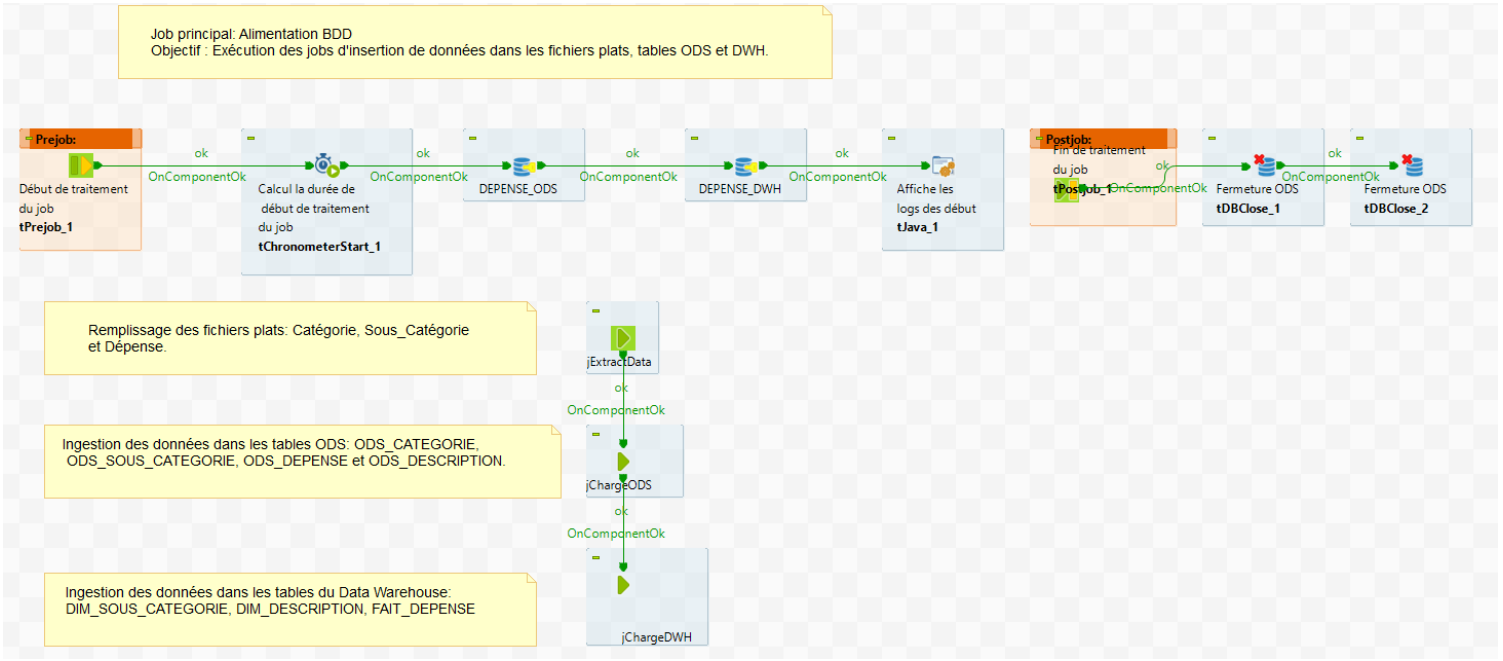
Un dernier job est nécessaire pour faire appel à tous les jobs créés auparavant, il sera nommé jAlimentation BDD. À son exécution, le job doit créer et charger les jobs :

jExtractData : qui extrait les données de fichier Excel et charge les fichier plats

jChargeODS : qui charge les tables ODS

jChargeDWH : qui charge les tables des dimensions du Data WareHouse

Ci-dessous un aperçu de l'ordonnancement du job Alimentation BDD :



Les jobs ont été exécutés avec succès dans Talend Studio. La vérification des résultats s'est faite dans PostgreSQL en exécutant les requêtes ci-dessous.

Remplissage de la table DIM_SOUS_CATEGORIE :

- Nombres de lignes de la table Dim_Sous_Categorie

```
SELECT count(*) FROM "DEPENSE_DWH"."DIM_SOUS_CATEGORIE";
```

	count bigint
1	24

- Les cinq premières lignes de la table Dim_Sous_Categorie :

```
SELECT * FROM "DEPENSE_DWH"."DIM_SOUS_CATEGORIE" LIMIT 5;
```

ID_DIM_SOUS_CATEGORIE [PK] integer	LB_SOUS_CATEGORIE character varying (100)	LB_CATEGORIE character varying (100)	DT_INSERTION timestamp without time zone
7	Abonnement Logiciel	Dépenses Fixes	2024-06-04 18:16:34.353
8	Abonnement Télécommunications	Dépenses Fixes	2024-06-04 18:16:34.353
9	Abonnement Transport	Dépenses Fixes	2024-06-04 18:16:34.353
10	Assurance Maison	Dépenses Fixes	2024-06-04 18:16:34.353
11	Assurance Voiture	Dépenses Fixes	2024-06-04 18:16:34.353

Remplissage de la table DIM_DESCRIPTION :

- Nombres de lignes de la table DIM_DESCRIPTION

```
SELECT count(*) FROM "DEPENSE_DWH"."DIM_DESCRIPTION";
```

	count bigint
1	24

- Les cinq premières lignes de la table Dim_Description

```
SELECT * FROM "DEPENSE_DWH"."DIM_DESCRIPTION" LIMIT 5;
```

	ID_DIM_DESCRIPTION [PK] integer	LB_DESCRIPTION_DEPENSE character varying (255)	DT_INSERTION timestamp without time zone	LB_JOB_NAME character varying (100)
1	1	Paiement Facture d'eau	2024-06-04 12:56:42.628	jDimDescription
2	2	Frais pharmacie	2024-06-04 12:56:42.629	jDimDescription
3	3	Paiement assurance voiture	2024-06-04 12:56:42.629	jDimDescription
4	4	Frais santé	2024-06-04 12:56:42.629	jDimDescription
5	5	Frais bancaire	2024-06-04 12:56:42.629	jDimDescription

Remplissage de la table DIM_CALENDAR :

- Nombres de lignes de la table DIM_CALENDAR

```
SELECT count(*) FROM "DEPENSE_DWH"."DIM_CALENDRIER";
```

	count bigint
1	2192

- Les cinq premières lignes de la table DIM_CALENDAR

```
SELECT * FROM "DEPENSE_DWH"."DIM_CALENDRIER" LIMIT 5;
```

ID_DIM_CALENDRIER [PK] integer	DT_JOUR date	NB_ANNEE integer	NB_SEMESTRE integer	NB_TRIMESTRE integer	NB_MOIS integer	NB_JOUR_ANNEE integer	NB_JOUR_MOIS integer	NB_JOUR_SEMAINE integer
20200101	2020-01-01	2020	1	1	1	1	1	3
20200102	2020-01-02	2020	1	1	1	2	2	4
20200103	2020-01-03	2020	1	1	1	3	3	5
20200104	2020-01-04	2020	1	1	1	4	4	6
20200105	2020-01-05	2020	1	1	1	5	5	7

Remplissage de la table FAIT_DEPENSE :

- Nombres de lignes de la table Fait_Depense

```
SELECT count(*) FROM "DEPENSE_DWH"."FAIT_DEPENSE";
```

	count bigint
1	357

- Les cinq premières lignes de la table Fait_Depense :

```
SELECT * FROM "DEPENSE_DWH"."FAIT_DEPENSE" LIMIT 5;
```

	ID_FAIT_DEPE [PK] integer	ID_DIM_CALENDRIER integer	ID_DIM_DESCRIPTION integer	ID_DIM_SOUS_CATEGORIE integer	NUM_DEPENSE character varying	NB_MT_DEPENSE numeric	NB_DEPENSE integer	DT_INSERTION timestamp without time zone
1	1	20220101	14	19	D00001	32.41	1	2024-06-04 22:07:57.505
2	2	20220102	18	18	D00002	20.37	1	2024-06-04 22:07:57.506
3	3	20220106	24	7	D00003	6.48	1	2024-06-04 22:07:57.506
4	4	20220109	18	18	D00004	20.82	1	2024-06-04 22:07:57.506
5	5	20220111	14	19	D00005	32.41	1	2024-06-04 22:07:57.506

Planification du job jAlimentationBDD

Étant donné que les données sources évoluent en permanence, les jobs doivent être planifiés de manière régulière. Par exemple, leur exécution peut être programmée chaque nuit afin de prendre en compte les modifications apportées durant la journée. Cette planification peut être réalisée à l'aide du planificateur de tâches du système d'exploitation.

Travail à réaliser :

– Exporter chaque job dans le répertoire C:/orion.
  - On clic droit sur le job puis sur construire le job
  - On sélectionne le type de construction : job standalone
  - On coche l'option Extract the zip file
  - Options par défaut puis finish

– On relance les jobs en cliquant sur les fichiers .bat.
– On vérifie que tout a bien fonctionné. Les jobs vont maintenant être planifiés grâce au planificateur de tâches de Windows.

Travail à réaliser :

– Ouvrir le planificateur de tâches de Windows
  - Accessoires / Outils système / Planificateur de tâches

– On clique sur créer une tâche… et l'on donne un nom : RunJobAlimentationBDD

- On crée un déclencheur (dans 15 min par exemple)
- On crée une action :
    - Action: Démarrer un programme
    - Programme :
        - C:\BI\DEPENSE\BUILD\JAlimentation\JAlimentation_run.bat
- Attendre que la tâche se lance.
- Vérifier que tout a bien fonctionné.

Conclusion

Le système d'extraction mis en place permet de récupérer les données à partir d'un fichier Excel et de générer trois fichiers plats. Ces fichiers sont créés dans un répertoire daté du jour, d'où la mise en place de répertoires horodatés pour chaque exécution.

À partir de ces trois fichiers horodatés, nous avons pu alimenter l'ODS (Operational Data Store). Ensuite, les données issues des différentes tables de l'ODS ont permis d'alimenter l'entrepôt de données (Data Warehouse).

Pour automatiser l'ensemble du processus, nous avons créé un job principal nommé *jAlimentationBDD*. Ce job regroupe les différentes étapes d'alimentation et a été planifié à l'aide du planificateur de tâches de Windows. Ainsi, l'exécution du job *jAlimentationBDD* est automatisée chaque jour à l'heure programmée, assurant une mise à jour régulière du Data Warehouse.