Conservatoire National des Arts et Métiers
Pascal AUREGAN

**Rapport de projet**

# NFE211

**Test de chaine décisionnelle sur cas simple.**
**Base opérationnelle en fichier et PostgresSQL**
**ETL avec TALEND Data Integration,**
**Reporting avec SAS**

le cnam

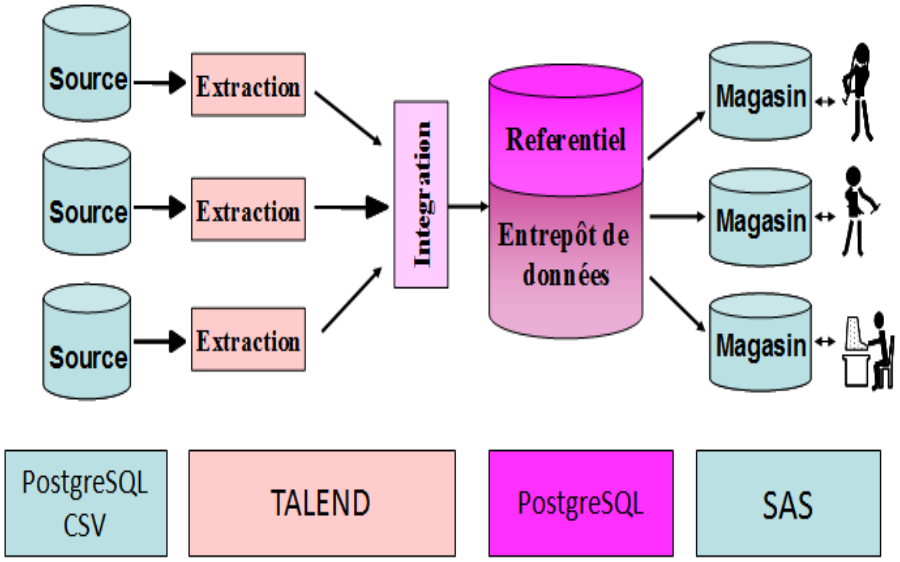
# Table des matières

# 1 Introduction

Le but de se projet est de mettre en place une chaine décisionnelle. L'objectif de ce projet n'étant pas de pousser dans ses retranchements les outils de la chaine décisionnelle, il a été décidé de créer de toute pièce un jeu de données. Le thème de ce jeu de données est l'achat de BD comme dans le cours. La base opérationnelle décrit les factures d'un ensemble librairies en France dont les clients sont connus. La chaine décisionnelle doit être en mesure de répondre in fine à la question : analyser les ventes de BD en France en fonction des clients, du lieu et date d'achat. La couche opérationnelle est assurée par une base de données de fichier csv et une base de données PostgresSQL.

Pour la couche ETL, il a été choisi TALEND Data Integration.

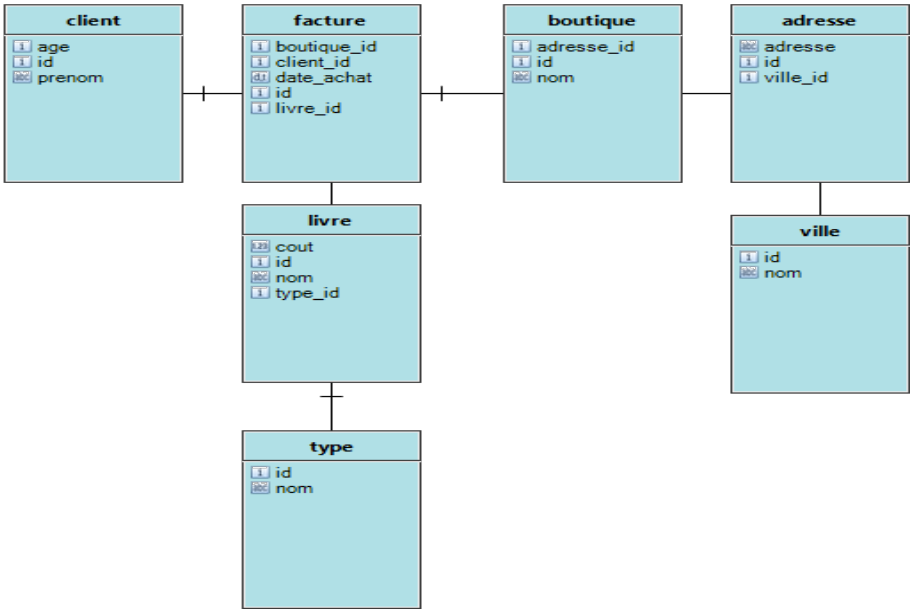
La couche reporting étudiée lors de ce projet est assurée par SAS Studio Version Etudiante.



# 2 Jeu de données de la base OLTP

## 2.1 Modèle relationnel

### 2.1.1 Présentation




Le modèle relationnel est un modèle normalisé.

— Une facture a été établie pour un client donné dans une boutique donnée à une date donnée et en rapport avec un livre donné.

— Le livre appartient à un type donné (Roman, BD, etc.).

— La boutique a une adresse située dans une ville

Voici combien de lignes ont chaque table :

— adresse : 26

— boutique : 25

— client : 8819

— facture : 8819

— livre : 8

— type : 3

En annexe A.1 , le code R qui a servi à générer les données dont il a été estimé que la pertinence n'avait que peu d'importance pour l'exercice. De plus, nous possédons un fichier de type csv contenant un dictionnaire de villes avec les départements associés ainsi que les régions. Voici en exemple les quatre premières lignes de ce fichier villes_def.csv

```
"ville","departement","region_name"
"Beauvais","Oise","Picardie"
"Compiegne","Oise","Picardie"
"Clermont","Oise","Picardie"
```

# 3 TALEND Open Studio for Data Integration 5.6

## 3.1 Création du modèle OLAP

### 3.1.1 Question à répondre

Analyse des ventes de bande dessinées d'une enseigne de librairie possédant plusieurs boutiques dans plusieurs villes.

### 3.1.2 Modèle ROLAP

L'implémentation ROLAP a été choisie par rapport au modèle MOLAP car même si la quantité de données était petite, il m'a semblé important d'étudier ce modèle qui paraît être très utilisé. Ne voyant rien justifier un modèle en flocon permettant de gagner de l'espace de stockage, j'ai utilisé un modèle en étoile. La table de faits est la table vente. Les dimentsions sont le temps, la boutique(lieu d'achat), le client, et le livre. L'étoile est de type transaction.

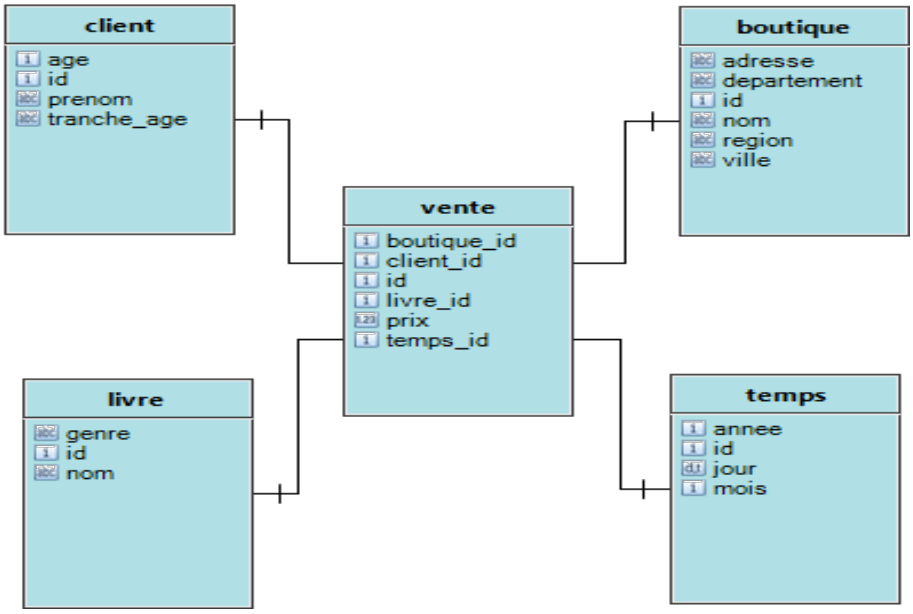



**Table de faits vente**

**Indicateurs** prix de la vente ; ce que le vente a rapporté

**fonctions d'agrégat**

— addition sur le prix

— moyenne sur le prix

**Dimension client**

— id

— prenom

— age

— tranche_age ("-12", "12-25", "35-65", "25-34", "+65")

**hiérarchie** age $<$ tranche_age

**Dimension boutique**

— id

— nom

— adresse

— departement

— ville

— region

**hiérarchie** adresse < ville < department < region

**Dimension livre**

— id
— nom
— genre

**hiérarchie** nom < genre

**Dimension Temps**

— id
— jour
— mois
— annee

**hiérarchie** jour < mois < annee

## 3.2 Intégration avec TALEND

### 3.2.1 Téléchargement Installation

Le téléchargement se fait sur http://fr.talend.com/download/data-integration. La documentation de l'outil est disponible au même endroit dans la partie Manuels d'utilisateurs et est facilement accessible. L'outil est en fait basé sur Eclipse ce qui le rend facilement appréhendable pour ceux qui connaissent le célèbre IDE mais vient aussi avec ses défauts.

### 3.2.2 Méthode utilisée

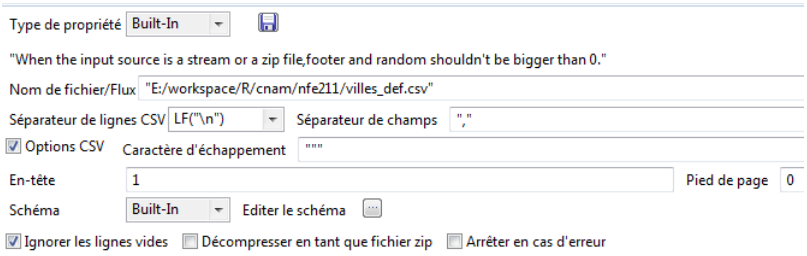
Pour chaque dimension, j'ai essayé d'avoir un cas d'usage différent :

— La dimension livre se construit par une jointure classique entre deux tables.
— La dimension client est construite en utilisant la table client suivie d'une transformation sur l'un de ses champs afin de déduire la tranche d'age à partir de l'age.
— La dimension boutique utilise des sources de données de type différents (CSV et postgres).
— La dimension temps construite à partir de la table temps subit un action sur ses données afin d'enlever les doublons et de déterminer les hiérarchies à partir d'une date.

### 3.2.3 Configuration des sources de données

Talend doit se connecter à la base de données postgres en lecture d'une part pour lire les données des tables de la base de données OLTP, d'autre part pour écrire en base dans la base de données de type ROLAP.


La source CSV est toute aussi facile à configurer. Il suffit de configurer le séparateur, la présence ou non d'une entête et les colonnes à prendre en compte comme le montre la figure ci-dessous :



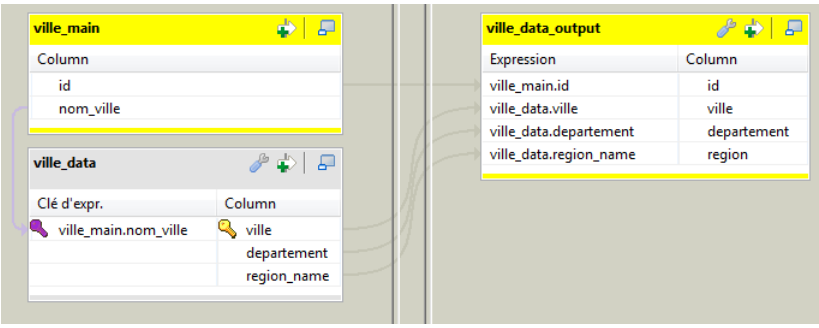

### 3.2.4 Dimension Boutique

#### Difficulté testée

La difficulté ici est de lier des données entre deux sources de données de type différents : CSV et Database.

#### Mise en œuvre

Les deux sources de données configurées nous permettent de faire une jointure simple avec le composant tMap. Le lien entre les entités se fait de manière intuitive sur cet écran :



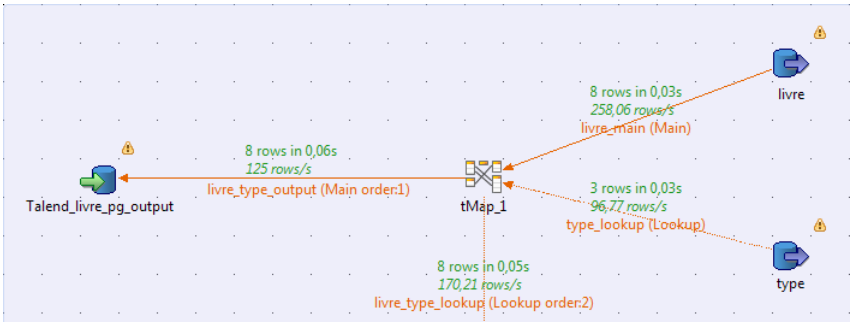

Dans la suite de ce projet nous utiliserons toujours une tMap pour faire une jointure entre deux sources de données.
Nous pouvons conclure que TALEND ne fait pas de distinction entre les différents types de données pourvu qu'elle soit composée de colonnes.

### 3.2.5 Dimension Livre

#### Difficulté testée

Aucune, jointure simple entre deux tables d'une base de données.

#### Mise en œuvre

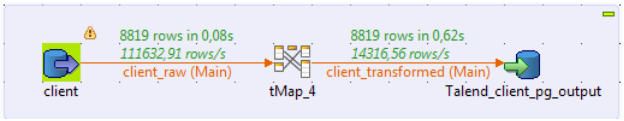



### 3.2.6 Dimension Client

#### Difficulté testée

La dimension client est construite en utilisant la table client suivie d'une transformation sur l'un de ses champs afin de déduire la tranche d'age à partir de l'age.

### Mise en œuvre



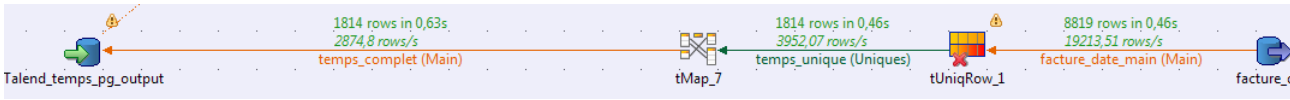

Pour transformer un age en tranche d'age, nous avons besoin de créer une routine dont le code est montré en annexe A.2 .

## 3.2.7 Dimension Temps

### Difficulté testée

La dimension temps construite à partir de la table temps subit un action sur ses données afin d'enlever les doublons et de déterminer les hiérarchies à partir d'une date.

### Mise en œuvre




Le composant tUniqRow_ que nous voyons sur ce schéma sert à enlever les doublons. C'est à dire que nous voulons une seule date dans notre dimension temps. Nous prenons donc l'ensemble des dates distinctes trouvées dans la table facture. Une fois ces dates trouvées, nous voulons déterminer les hiérarchies à partir du champs date_achat de la table facture. Nous utilisons pour cela les fonctions built-in de TALEND :

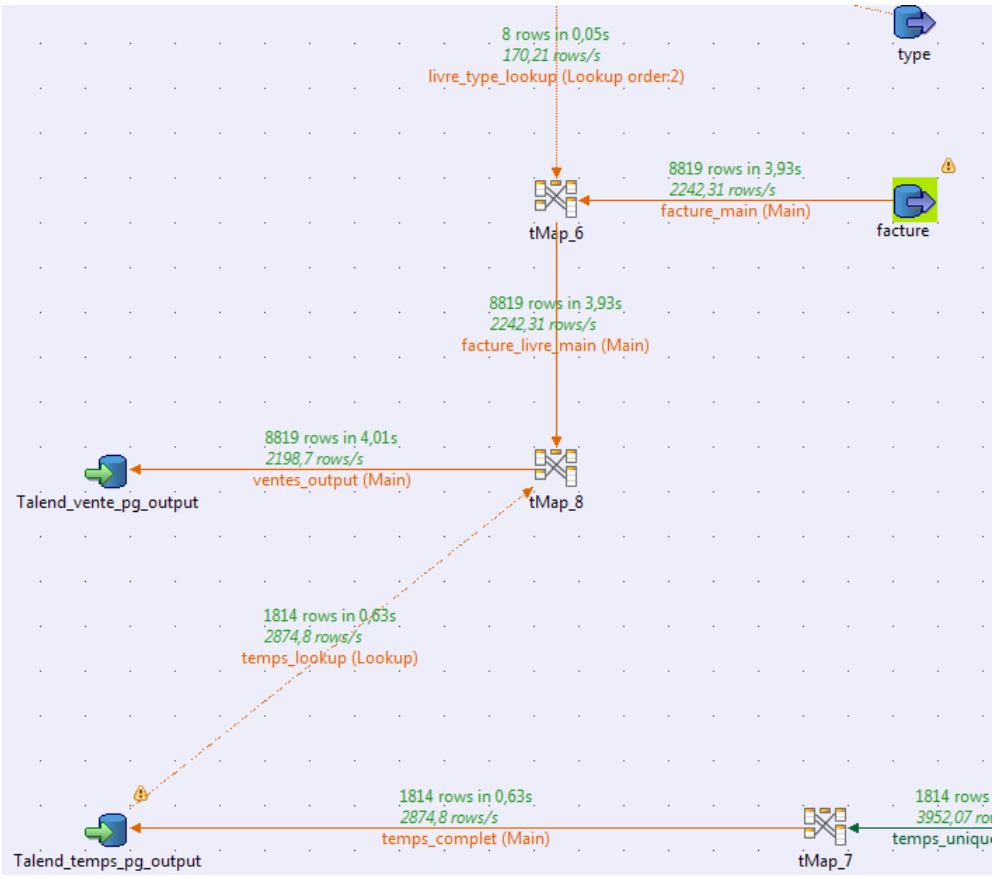
- TalendDate.getPartOfDate("MONTH", date_achat) pour déterminer le mois d'une date
- TalendDate.getPartOfDate("YEAR", date_achat)

## 3.2.8 Table de faits vente

### Difficulté testée

Aucune, l'intégrité relationnelle des données est gérée côté base OLTP, ce qui réduit la création de la table de faits à un ensemble de jointure.

**Mise en œuvre**




### 3.2.9 Conlusion sur l'utilisation de TALEND

Pour les cas testés qui paraissent classiques, TALEND se montre plutôt efficace. La montée en charge reste à tester. Le nombre de composant paraît énorme et je n'ai pas réussi à les utiliser tous. C'est d'ailleurs un des défauts de TALEND à mon sens. L'utilisation n'est pas très intuitive et la résolution de problème est très compliquée. Lorsqu'un problème survient, l'utilisateur est gratifié d'un NullPointerException sans plus d'information. La modification du code JAVA généré est anecdotique selon moi même si son utilité est révélée lorqu'il s'agit de débugger.

# 4 Reporting

## 4.1 SAS

### 4.1.1 Présentation

### 4.1.2 Processus d'intégration des données et base de données associée

### 4.1.3 Processus de maintenance de la base

# 5 Conclusion

Bonne chance

# A Annexes

## A.1 Code R ayant généré les données

Listing A.1 – generateData.R

```
library(dplyr)
library(tidyr)

setwd('E:/workspace/R/cnam/nfe211')

prenoms <- read.csv('liste_des_prenoms_par_annee.csv', sep = ';')


head(prenoms)
hist(prenoms$nombre)
summary(prenoms$nombre)
hist(x = prenoms$annee)
nb_clients <- length(prenoms$prenoms)


X <- 10 + rnorm(nb_clients, mean=5, sd = 1)
Y <- 5 + rnorm(nb_clients, mean=20, sd = 2)

U <- rbinom(n=nb_clients, size=1, prob=0.6)
summary(U)


hist(X)
```

```
hist(Y)
Z <- U * X + ((U-1)*-1 + Y)
hist(Z)
age <- floor(Z)
clients <- cbind(prenoms, age)
head(clients)
hist(clients$age)


type<-floor(1+rnorm(nb_clients, mean=1.4, sd=0.6)%%3)
hist(type)
clients <- cbind(clients, type)

livres <- c(6, 7, 1, 2, 4, 8, 3, 5)

# distribution sur livres
livres_achetes <- floor(rnorm(nb_clients, mean = 3.5, sd=1.5))
livres_achetes <- sapply(livres_achetes, FUN = function(x) max(0,x))
livres_achetes <- sapply(livres_achetes, FUN = function(x) min(7,x))
hist(livres_achetes)


clients <- cbind(clients, livres_achetes)
head(clients)

# lien vers la table livre
livre <- sapply(livres_achetes, FUN=function(x)livres[x+1])
clients <- cbind(clients, livre)


hist(clients$livre)

boutiques <- floor(runif(nb_clients, min=1, max=25))
clients <- cbind(clients, boutiques)
hist(boutiques)

client_id <- 1:length(clients$prenoms)
```

```
clients <- cbind(clients, client_id)

#last date in the data
end_date <- as.Date('2015-01-01')

date_achat <- end_date - runif(n=nb_clients, max=365*5, min=1)



clients <- cbind(clients, date_achat)
clients <- mutate(clients, prenom = prenoms)

clients_for_csv <- select(clients, id=client_id, age, prenom)
write.csv(x=clients_for_csv, file='clients.csv', row.names=FALSE)
nb_clients + 1 - client_id

clients <- mutate(clients, facture_id = (nb_clients + 1 - client_id))

facture_for_csv <- select(clients, id=facture_id, date_achat, boutique_id=b
write.csv(x=facture_for_csv, file='factures.csv', row.names=FALSE)

villes <- read.csv(file='ville.csv')
departements <- read.csv('departement.csv')
regions <- read.csv('region.csv')
```

## A.2 Routine java toTrancheDage.java

Listing A.2 – toTrancheDage.java

```
public class to_tranche_age {
        private enum TRANCHE_AGE{
                LT_12("-12"),
                B12_25("12-25"),
                B25_34("25-34"),
                B35_65("35-65"),
                MT_65("+65");

                TRANCHE_AGE(String label){
                        this.label = label;
                }

                private final String label;

        }

    public static String perform(int age){
        if(age < 0){
                throw new IllegalArgumentException(
                        "age should not be less than 0: " + age);
        }
        if(age < 12){return TRANCHE_AGE.LT_12.label;}
        if(age >= 12 && age <= 25){return TRANCHE_AGE.B12_25.label;}
        if(age > 25 && age <= 34){return TRANCHE_AGE.B25_34.label;}
        if(age > 34 && age <= 65){return TRANCHE_AGE.B35_65.label;}
        return TRANCHE_AGE.MT_65.label;
    }
}
```