

Document de conception – Projet ODE

|  |  |  |
| --- | --- | --- |
| **Suivi des versions** | | |
| **Date** | **Version** | **Objet de la version** |
| **08/07/2015** | 01 | [Olivier] Création du document |
| **23/07/2015**  **26/07/2015**  **26/07/2015**  **23/07/2015**  **21/07/2015** | 02  02  03  02  02 | [Bernard] : Relecture OK  [Brice] : Relecture OK  [Cédric] : Relecture OK  [Olivier] : Relecture OK  [Thomas] : Relecture OK |
| **04/08/2015** | 03 | [Olivier] Correction suite à modifications de DIM\_TEMPS |
|  |  |  |

Sommaire

[Introduction 3](#_Toc424573269)

[Aspects métier 3](#_Toc424573270)

[Quelques chiffres 3](#_Toc424573271)

[Structure organisationnelle 3](#_Toc424573272)

[Architecture de la chaine décisionnel 3](#_Toc424573273)

[Vue d’ensemble 3](#_Toc424573274)

[Partie ETL 3](#_Toc424573275)

[Partie DWH / Cubes 3](#_Toc424573276)

[Partie Reporting 3](#_Toc424573277)

[Base opérationnelle 3](#_Toc424573278)

[Vue d’ensemble 3](#_Toc424573279)

[Table UNIVERS\_PRODUITS 3](#_Toc424573280)

[Table RAYONS\_PRODUITS 3](#_Toc424573281)

[Table FAMILLES\_PRODUITS 3](#_Toc424573282)

[Table SOUS\_FAMILLES\_PRODUITS 3](#_Toc424573283)

[Table PRODUITS 3](#_Toc424573284)

[Table CLIENTS 3](#_Toc424573285)

[Table LIEUX 3](#_Toc424573286)

[Table VILLES 3](#_Toc424573287)

[Table STOCKS 3](#_Toc424573288)

[Table VENTES 3](#_Toc424573289)

[Table TICKETS 3](#_Toc424573290)

[Table PRIXPRODUITS 3](#_Toc424573291)

[Entrepôt de données 3](#_Toc424573292)

[Vue d’ensemble 3](#_Toc424573293)

[Table de dimension CATEGORIES 3](#_Toc424573294)

[Table de dimension PRODUITS 3](#_Toc424573295)

[Table de dimension TEMPS 3](#_Toc424573296)

[Table de dimension LIEUX 3](#_Toc424573297)

[Table de dimension CLIENTS 3](#_Toc424573298)

[Table de dimension VILLES 3](#_Toc424573299)

[Table de faits VENTES 3](#_Toc424573300)

[Table de faits STOCKS 3](#_Toc424573301)

# Introduction

Le projet « Optimisation des Données de l’Entrepôt (ODE) » consiste à utiliser les techniques mathématiques vues dans le Master 2 MIAGE afin de construire l'entrepôt de données (Data-Warehouse - DWH) de manière optimal, en termes de temps de réponse à l’interrogation des cubes et d’occupation disque.

# Aspects métier

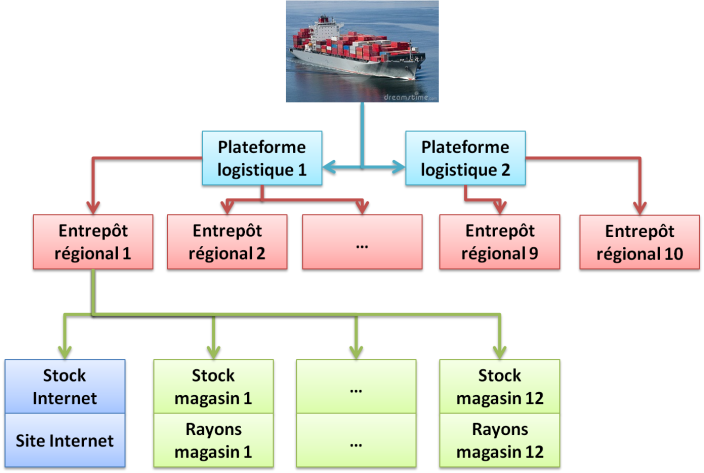
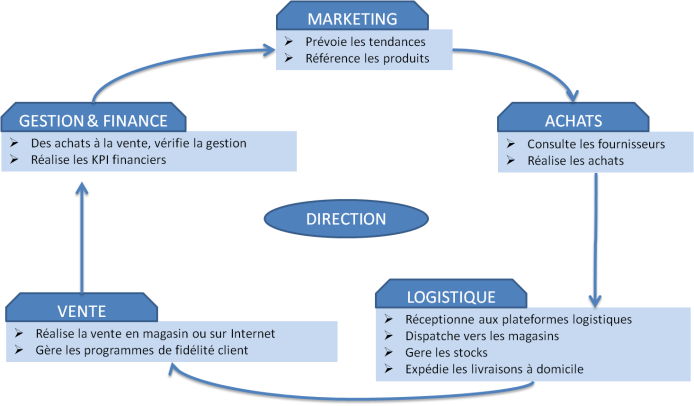
CASTO-MERLIN : Grande distribution de bricolage



## Quelques chiffres

* 31 300 collaborateurs
* 2 plateformes logistiques nationales
* 10 centrales de distribution régionales
* 121 magasins + 1 site Internet
* 60 000 références magasins plus 20 000 « sur commande »
* 3 856 fournisseurs

## Structure organisationnelle



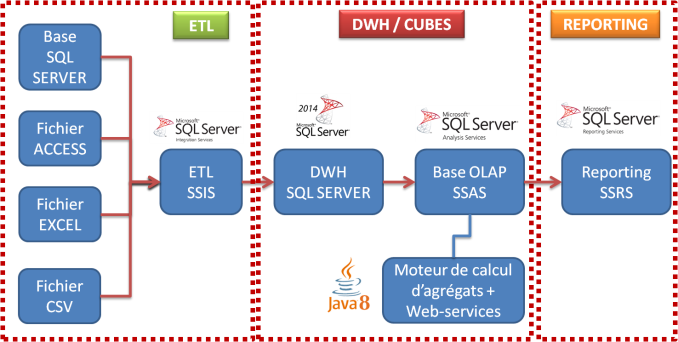
# Architecture de la chaine décisionnel

## Vue d’ensemble

Les données opérationnelles de l’entreprise, c'est-à-dire celle générées et utilisés pour la gestion de l’activité courante de l’entreprise, sont stockées principalement dans une base OLTP SQL Server 2014. Nous la désignerons par « base opérationnelle » dans la suite de ce document.

D’autres sources de données sont possibles :

* Fichiers « plats » type CSV
* Fichiers EXCEL
* Fichiers ACCESS



## Partie ETL

L’ETL sera en charge de collecter les informations à partir des différentes sources de données opérationnelles de l’entreprise :

* Base opérationnelle
* Fichiers CSV
* Fichiers EXCEL
* Fichiers ACCESS

Cette partie sera réalisée avec **SQL Server 2014 – Integration Services (SSIS)**

## Partie DWH / Cubes

C’est le « cœur » de la chaine décisionnelle que nous allons réaliser.

Le Datawarehouse (DWH – Entrepot de données) est modélisé en flocon, et hébergé sur une base OLTP **SQL Server 2014.**

Le cube sera hébergé sur une base OLAP **SQL Server 2014 Analysis Service (SSAS)**

Le moteur de calcul d’agrégats est un programme Java servant à « optimiser » la structure et le calcul des cubes (Agrégats, cf. cours du D111)

## Optimiseur - Bouton 1 : Algorithme de Métropolis

## Objectif

Le but de ce bouton est d'envoyer un design d'agrégats vers SSAS, calculé à partir de la méthode de l'algorithme de Métropolis.

## Etapes

### Récupération du treillis de cuboïdes

La première étape de cette fonction consiste à lister l'ensemble des combinaisons possibles, c'est à dire l'ensemble des cuboïdes potentiels à stocker. Il s'agit donc de trouver, à partir de nos tables n de dimensions, l'ensemble des 2n-1 (nous ne sommes pas intéressés par la solution nulle) combinaisons possibles.

Exemple : avec 3 dimensions (Temps, Clients, Produits) :

* Temps
* Clients
* Produits
* Temps \* Clients
* Temps \* Produits
* Clients \* Produits
* Temps \* Clients \* Produits

Pour cela, on utilise un algorithme récursif qui parcours la liste des dimensions en ajoutant à chaque récursion le préfix de la dimension par laquelle la fonction est rappelée. Cette fonction permet donc de renvoyer une liste d'objet Dimension (cf. partie Olivier) dont certains attributs sont complétés : Nom et Dimension (1-D, 2-D etc.).

### Calcul de la taille des cuboïdes

Le second élément décisif pour notre fonction est de connaître la taille potentielle des différents cuboïdes puisque comme nous le verrons plus tard, c'est cet élément qui déterminera d'une part la condition d'arrêt et d'autre part la performance de notre solution choisie.

Les Tailles (en octet) d'une ligne des dimensions 1-D et le Nombre de lignes des dimensions sont déjà récupérés par l'algorithme d'Olivier. Nous pouvons donc nous servir de ces éléments pour compléter les attributs de nos cuboïdes : il suffit donc de récupérer le nombre de lignes de chaque cuboïde et de le multiplier par la taille estimé d'une ligne du cuboïde pour connaitre son poids.

On considère que le poids d'une ligne d'un cuboïde est égale à la somme des poids d'une ligne des dimensions le composant.

Exemple : poids d'une ligne du cuboïde (Temps \* Clients) = poids\_1\_ligne(Temps) + poids\_1\_ligne(Clients).

Pour récupérer le nombre de ligne que contiendra un cuboïde, nous effectuons une requête MDX, le langage de requête pour les bases de données OLAP, qui croise les tables de dimension que contient le cuboïde avec notre table de faits et nous renvoie le résultat sous forme d'un tableau croisé. Il suffit donc ensuite de compter le nombre de ligne de notre tableau pour estimer le nombre de ligne de notre cuboïde.

Nous pouvons ensuite affecter ces variables aux attributs Taille d'une ligne et Nombre de lignes de notre liste d'objets Dimension.

Nous disposons donc enfin de tous les éléments nécessaires à l'algorithme de Métropolis.

### Application de l'algorithme de Métropolis à la recherche d'une solution de matérialisation d'un datacube.

Cette partie a pour objet de modéliser la recherche d'une solution acceptable via l'algorithme de Métropolis, adapté à notre projet.

#### Contexte

Partant de notre datawarehouse il s'agit de savoir quels cuboïdes nous allons précalculer pour améliorer l'efficience de nos requêtes lors de la création des rapports.

Sachant que :

* Si nous ne calculons aucuns cuboïdes, l'espace de stockage sera optimale mais l'évaluation des requêtes ne le sera pas.
* Si nous les calculons tous, l'évaluation des requêtes sera optimale mais l'espace de stockage sera saturé.

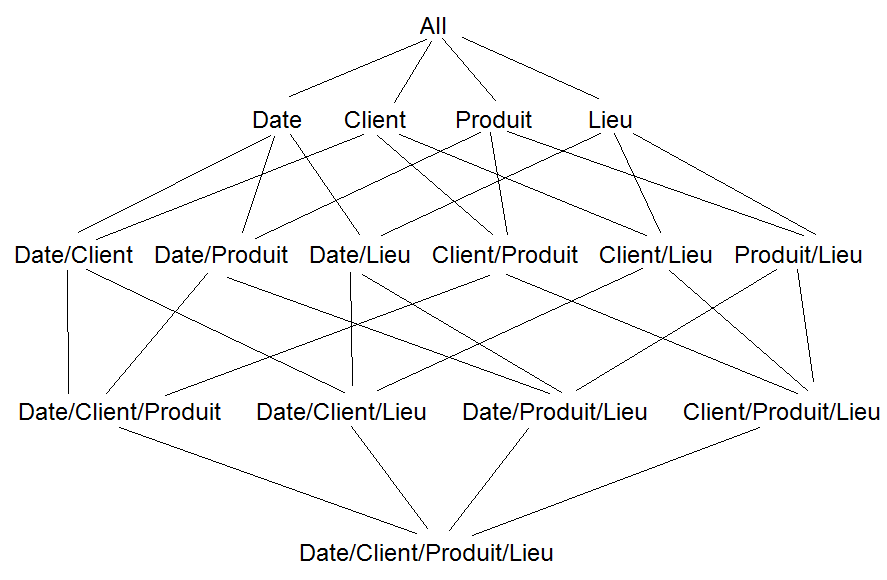
Il s'agit donc de trouver un compromis entre ces deux solutions.

#### Modélisation du problème

On décide de représenter notre problématique à l'aide d'une chaîne de Markov. Pour cela nous définissons une variable aléatoire Xn représentant la solution de stockage choisis à l'instant n. Les états de la chaine représente donc les différentes solutions de stockage qui existe pour notre datacube.

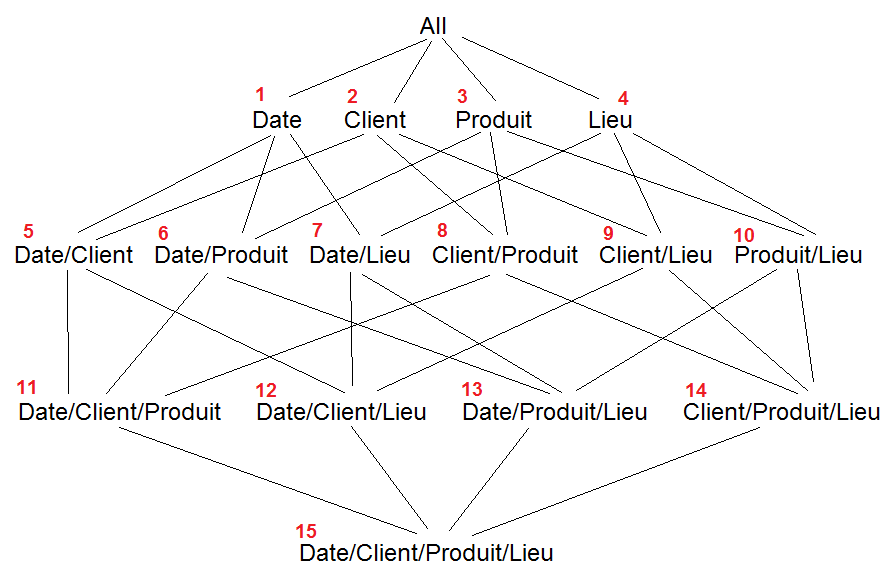
**Exemple pour mieux comprendre :**

Schéma simplifié de notre datacube :



Nous avons donc 15 cuboïdes potentiellement intéressant à stocker (le All représente notre datawarehouse que l'on doit forcément stocker pour pouvoir calculer toutes les requêtes).

Numérotons ces cuboïdes :

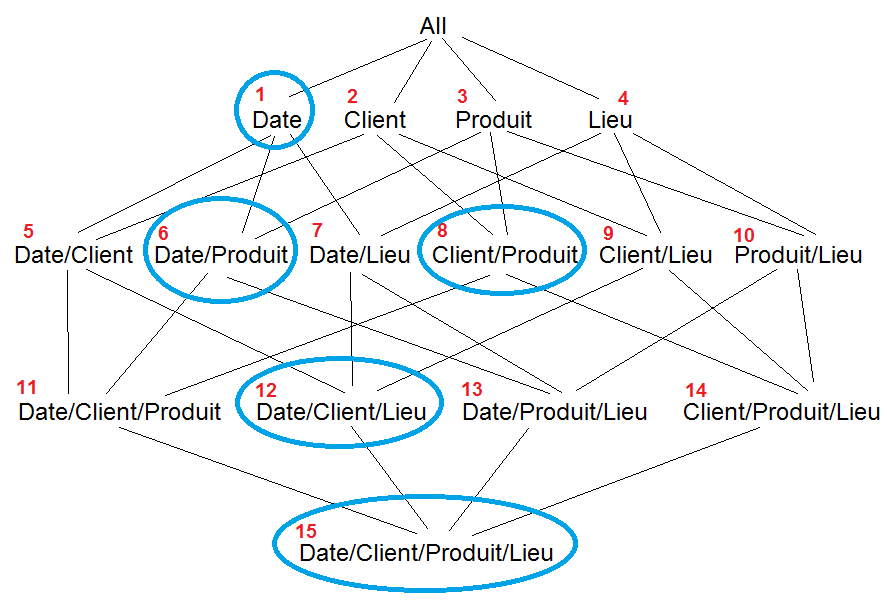


Un état de notre chaîne, c'est à dire une possibilité de solution de stockage peut donc être représenté par le fait que certains cuboïdes sont stockés (valeur 1 par exemple) et les autres non (valeurs 0).

**Exemple d'une solution de stockage :**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Stocké (1) / Non stocké (0)** | 1 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | 1 |
| **N° du cuboïde** | **1** | **2** | **3** | **4** | **5** | **6** | **7** | **8** | **9** | **10** | **11** | **12** | **13** | **14** | **15** |

Ce qui correspond sur notre schéma à :



Dans cette solution, les cuboïdes entourés de bleu seraient donc stockés et pas les autres.

Cette solution seraient donc en base 2 : 100001010001001 soit en base 10 : 17 033.

Il y aurait donc au total pour notre schéma de datacube ci-dessus 215 possibilités de stockage différentes, soit 32 768 états (en comptant la solution 0 qui consiste à ne rien stocker) et donc une matrice de 32 768 x 32 768.

#### Passage aux matrices

Intéressons nous maintenant aux probabilités de transitions.

On modélise une transition par l'action de choisir un cuboïde parmi les 15 et de changer son statut (de stocké à non stocké ou inversement). Ainsi, tous les états sont accessibles depuis un autre états par une série de permutation successive du statut des cuboïdes.

**Exemple :**

On peut passer de l'état (i.e. la solution) 100001010001001 à 000001010101011 en faisant :

100001010001001 -> **0**00001010001001 -> 000001010**1**01001 -> 0000010101010**1**1

Les probabilités de transitions non nulles sont égales à 1/15 : 1 chance sur 15 de choisir un cuboïde et de permuter son statut.

**Exemple :**

Depuis la solution 0 (en base 2 : 00000000000000), on peut passer aux solutions :

* 1 (000000000000001)
* 2 (000000000000010)
* 4 (000000000000100)
* 8 (000000000001000)
* 16 (000000000100000)
* ...
* 16 384 (100000000000000)

Toutes les autres probabilités de transition depuis l'état 0 étant égales à 0. On ne peut donc pas passer (directement) de l'état 0 à l'état 3 par exemple.

Début de notre matrice de transition :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **États** | **0** | **1** | **2** | **3** | **4** | **5** | … |
| **0** | **0,00** | 0,07 | 0,07 | 0,00 | 0,07 | 0,00 | … |
| **1** | 0,07 | **0,00** | 0,00 | 0,07 | 0,00 | 0,07 | … |
| **2** | 0,07 | 0,00 | **0,00** | 0,07 | 0,00 | 0,00 | … |
| **3** | 0,00 | 0,07 | 0,07 | **0,00** | 0,00 | 0,00 | … |
| **4** | 0,07 | 0,00 | 0,00 | 0,00 | **0,00** | 0,07 | … |
| **5** | 0,00 | 0,07 | 0,00 | 0,00 | 0,07 | **0,00** | … |
| … | … | … | … | … | … | … | **0,00** |

On remarque que cette matrice de transition est symétrique et irréductible (il existe un chemin de chaque état vers tous les autres).

**Exemple :** P(3,1) = P(1,3)

Nous avons donc créé une chaine qui permet en la simulant de choisir aléatoirement de manière uniforme une solution parmi toutes.

On va donc pouvoir utiliser l'algorithme de Métropolis.

#### Application de l'algorithme de Métropolis

Commençons par Métropoliser notre chaîne.

En effet, pouvoir choisir une solution de manière aléatoire n'est pas spécialement intéressant (un randint(32 768) ferait la même chose...) il faut donc influencer notre modèle pour que les solutions les plus efficientes aient plus de chance d'être sélectionnées que les solutions les moins efficientes.

Nous devons tout d'abord fixer une contrainte, car si l'on se contente d'évaluer l'efficacité des différentes solutions, on peut directement conclure que c'est la solution : 111111111111111 qui sera la plus efficace puisque celle-ci stocke tous les cuboïdes et fait donc et permet donc une évaluation des requêtes optimale.

La contrainte naturelle semble logiquement être la volumétrie des cuboïdes à stocker. A noter que l'on pourrait également choisir la contrainte du nombre de cube à stocker mais celle-ci semble avoir moins de sens logique que la volumétrie.

L'ajout de cette contrainte oblige à modifier légèrement notre chaine de Markov puisqu'en effet, certains état ne seront plus accessibles (les solutions trop volumineuses) et notre chaîne ne sera donc plus symétrique.

**Un exemple pour s'en convaincre :**

Reprenons le début de notre matrice précédente :

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **États** | **0** | **1** | **2** | **3** | **4** | **5** | … |
| **0** | **0,00** | 0,07 | 0,07 | 0,00 | 0,07 | 0,00 | … |
| **1** | 0,07 | **0,00** | 0,00 | 0,07 | 0,00 | 0,07 | … |
| **2** | 0,07 | 0,00 | **0,00** | 0,07 | 0,00 | 0,00 | … |
| **3** | 0,00 | 0,07 | 0,07 | **0,00** | 0,00 | 0,00 | … |
| **4** | 0,07 | 0,00 | 0,00 | 0,00 | **0,00** | 0,07 | … |
| **5** | 0,00 | 0,07 | 0,00 | 0,00 | 0,07 | **0,00** | … |
| … | … | … | … | … | … | … | **0,00** |

Imaginons que notre solution n°5 dépasse la contrainte de volumétrie : cet état ne doit donc plus être accessible. Les probabilités de transitions de la ligne 5 vont passer à 0 (aucun soucis) et celles de la colonne 5 également (là il y a un soucis). Or si l'on prend notre état 1, sa probabilité de transition d'aller vers l'état 5 est donc maintenant nulle : ses probabilités de transition deviennent donc de 1/14 pour ses 14 états restants accessibles (contre 1/15 auparavant) et restent/deviennent 0 pour ceux non accessibles (dont l'état 5). On voit qu'avec cette modification : P(1,3) = 1/14 <> P(3,1) = 1/15. Car l'état 3 n'a pas été impacté par la non-accessibilité de l'état 5.

La solution à ce problème consiste à ajouter la probabilité de transition P(1,5) à P(1,1) ainsi P(1,1) prendra la valeur 1/15 rétablissant la symétrie. Cette solution consiste tout simplement à considérer que lorsque l'on ne peut pas aller sur l'état qui ne satisfait pas la contrainte de volumétrie on reste sur l'état actuel.

Si l'on considère donc uniquement les états accessibles après application de la contrainte, notre matrice est bien symétrique et irréductible.

Il s'agit donc ensuite de définir la notion d'efficience pour notre solution.

Plusieurs solution peuvent être envisagées. La solution la plus efficiente peut être celle qui :

* Stocke le plus grand nombre de cuboïdes.
* Stocke la volumétrie totale des cuboïdes la plus importante.
* Permet une évaluation des requêtes optimale.

Tout en satisfaisant bien entendu la contrainte définit précédemment.

Quel que soit la solution retenue, nous modifierons notre fonction de transition pour que les probabilités de transition soient de la forme : p = min{v'(t) / v(t),1}

v' étant par exemple le nombre de cube stockés pour la solution de destination et v étant le nombre de cube stockés pour la solution actuelle. Ainsi, si v'>v, alors le ratio v'/v sera supérieur à 1 et donc min{v'/v,1} prendra la probabilité 1 : la transition se fera obligatoirement vers le nouvel état. Si v' < v, la transition ne se fera qu'avec probabilité v'/v : on aura donc tout de même une chance d'aller vers cette solution pour se donner la possibilité de visiter toutes les solutions possibles.

#### Conclusion & discussion

A noter qu'en réfléchissant bien, il est possible de résoudre algoriquemiquement notre problématique de manière plus simple que via l'utilisation des matrices. En effet, il "suffirait" de trier les différentes solutions selon le critère d'efficience choisit (nombre de cubes, évaluation des requêtes ou volumétrie de la solution) et de choisir celui qui à la plus forte valeur tout en ne dépassant pas le seuil de contrainte fixé.

Néanmoins, la technique des matrices devient plus intéressante dans le cas où l'on a un grand nombre de solutions à explorer. En effet, comme on a pu le voir dans notre exemple trivial avec seulement 4 dimensions, le nombre de solutions est de 32 768, or on sait d'ores et déjà que dans n'importe quel modèle de données utilisé dans un environnement réel le nombre de dimensions sera facilement décuplé. Exemple, pour 30 dimensions (ce qui se rapproche de notre ODE's Project) le nombre de solutions serait de 230 soit 1 073 741 824. Le nombre de dimensions d'un projet "réel" d'entreprise serait probablement bien plus élevé. L'algorithme de Métropolis tirera donc sa force du fait que l'on ne sera pas obligé de parcourir l'ensemble des solutions mais qu'en faisant tourner cet algorithme avec un nombre d'itérations assez important (i.e. supérieur à 100) on sera sûr d'obtenir une solution relativement bonne.

On pourrait malgré tout penser que les limitations techniques qui empêcherait de parcourir l'ensemble des dimensions s'appliqueront également à notre algorithme de Métropolis puisqu'en effet, la création et la manipulation de matrices 1 000 000 000 x 1 000 000 000 demanderait sans doute énormément de ressources. Cependant on voit qu'en programmant réellement l'algorithme, l'utilisation de matrice est facultatif et ne sert en fait finalement qu'en terme de modélisation du problème et de résolution (validation) de la solution. On ne se sert au final dans le code que des propriétés de notre modèle pour obtenir une solution viable.

#### Algorithme de la solution

**Exemple du pseudo-code utilisé :**

Note : on choisit comme indicateur de performance d'une solution la volumétrie totale de la solution. Un solution A étant meilleure qu'une solution B si poids(A) > poids(B). L'algorithme suivant est bien entendu adaptable a d'autres indicateur de performance.

**algo\_Metropolis :**

{

*// définition des variables*

*// initialisation de notre solution actuelle = un tableau de taille Nombre de cuboïdes*

sol\_act = [0,0,0,...,0,0,0]

*// le poids de stockage de chaque cuboïde*

poids\_cubes = [10,25,62,...,40,3,16]

*// le seuil de poids de la solution totale à ne pas dépasser*

seuil\_poids = 400

*// initialisation du poids de notre solution actuelle*

poids\_act = 0

*// initialisation du poids de la solution de destination (utile pour les tests)*

poids\_next = 0

*// initialisation du compteur*

i = 0

*// initialisation de la variable pour le choix du prochain cube à stoker/retirer*

choix\_cube = 0

*// début de la boucle principale*

**tant que** ( i < 1000 )

{

*// choix d'un cuboïde parmi les 15*

choix\_cube = randint(15)

*// on calcul le poids de la future solution en ajoutant/retirant le poids du cuboïde sélectionné*

**si** (sol\_act[choix\_cube] == 0)

poids\_next = poids\_act + poids\_cubes[choix\_cube]

**sinon**

poids\_next = poids\_act - poids\_cubes[choix\_cube]

// *si le poids de la future solution ne dépasse pas le seuil*

**si** poids\_next < ou = seuil\_poids

{

*// si la future solution est meilleure que la solution actuelle <=> v'/v > 1*

**si** poids\_next > ou égal poids\_act

{

*// on choisit la future solution*

sol\_act[choix\_cube] = 1

poids\_act = poids\_next

}

*// si la future solution est moins bonne que la solution actuelle*

**sinon**

{

*// on effectue un tirage de Bernoulli avec probabilité v'/v*

**si** tirage\_bernouilli(poids\_next/poids\_act) == 1

{

*// si tirage positif on va tout de même vers la moins bonne solution*

sol\_act[choix\_cube] = 0

poids\_act = poids\_next

}

}

}

i = i + 1

}

}

## Etape suivante

Il s'agit enfin de transformer notre solution retournée, suite de 0 et de 1 en une solution "compréhensible" par SSAS, c'est à dire un design d'agrégats. C'est l'objet de la partie suivante.

## Partie Reporting

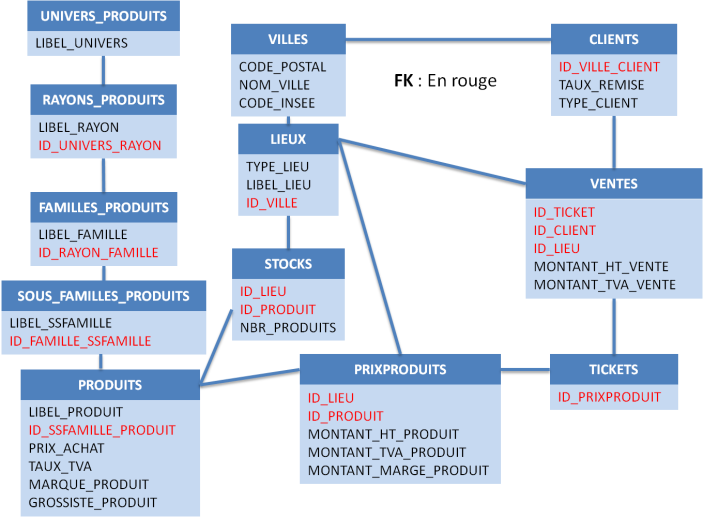
La chaine décisionnelle génère des reports sur un sujet « métier ». Par exemple : Calcul du CA, de la marge, évolution des ventes…

Cette partie sera réalisée avec **SQL Server 2014 – Reporting Services (SSRS)**

# Base opérationnelle

## Vue d’ensemble

Dans la base dédiée « **BaseOperationelleODE** », nous créons un schéma dédié « **ODE\_VENTES** », qui contient les tables suivantes :



## Table UNIVERS\_PRODUITS

But

Les Univers de produits sont les premiers niveaux de classification des produits vendus par CASTO-MERLIN.

Il comprend actuellement 3 valeurs :

* « Interieur Et Decoration »
* « Atelier Et Materiaux »
* « Jardin Et Exterieur »

Chaque libellé est associé à un ID numérique, qui sera utilisé en tant que clé primaire : Les libellés peuvent évoluer, mais pas leurs ID associés.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_UNIVERS | int | Non |
| LIBEL\_UNIVERS | nvarchar(256) | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table RAYONS\_PRODUITS

But

Les rayons de produits sont les seconds niveaux de classification des produits vendus par CASTO-MERLIN.

Il comprend actuellement 39 valeurs :

* « Chauffage, Climatisation Et Ventilation »
* « Cuisine »
* « Décoration »
* …

Chaque libellé est associé à un ID numérique, qui sera utilisé en tant que clé primaire : Les libellés peuvent évoluer, mais pas leurs ID associés. Chaque rayon est associé à exactement un Univers, au travers du champ ID\_UNIVERS\_RAYON, qui est une clé étrangère vers la table des Univers de produits.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_RAYON | int | Non |
| LIBEL\_RAYON | nvarchar(256) | Non |
| ID\_UNIVERS\_RAYON | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table FAMILLES\_PRODUITS

But

Les familles de produits sont les troisièmes niveaux de classification des produits vendus par CASTO-MERLIN.

Il comprend actuellement 317 valeurs :

* « Chauffage Électrique »
* « Chauffage Central »
* « Chauffage Bois Et Bois De Chauffage »
* …

Chaque libellé est associé à un ID numérique, qui sera utilisé en tant que clé primaire : Les libellés peuvent évoluer, mais pas leurs ID associés. Chaque famille est associée à exactement un rayon, au travers du champ ID\_RAYON\_FAMILLE, qui est une clé étrangère vers la table des rayons de produits.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_FAMILLE | int | Non |
| LIBEL\_FAMILLE | nvarchar(256) | Non |
| ID\_RAYON\_FAMILLE | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table SOUS\_FAMILLES\_PRODUITS

But

Les sous-familles de produits sont les quatrièmes niveaux de classification des produits vendus par CASTO-MERLIN.

Il comprend actuellement 1 212 valeurs :

* « Poêle À Pétrole »
* « Poêle À Gaz »
* « Radiateur Soufflant »
* …

Chaque libellé est associé à un ID numérique, qui sera utilisé en tant que clé primaire : Les libellés peuvent évoluer, mais pas leurs ID associés. Chaque sous-famille est associée à exactement une famille, au travers du champ ID\_FAMILLE\_SSFAMILLE, qui est une clé étrangère vers la table des familles de produits.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_SSFAMILLE | int | Non |
| LIBEL\_FAMILLE | nvarchar(256) | Non |
| ID\_RAYON\_SSFAMILLE | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table PRODUITS

But

Cette table contient les produits vendus par CASTO-MERLIN.

Chaque libellé de produit est associé à un ID numérique, qui sera utilisé en tant que clé primaire : Les libellés peuvent évoluer, mais pas leurs ID associés. Chaque produit est associé à exactement une sous-famille, au travers du champ ID\_ SSFAMILLE\_PRODUIT, qui est une clé étrangère vers la table des sous-familles de produits.

La table contient toutes les informations relatives au produit, indépendamment du circuit de vente :

* Description courte (LIBEL\_PRODUIT) et détaillée (DESC\_PRODUIT) du produit
* Prix d’achat HT du produit auprès du grossiste, ou directement du fabriquant
* TVA applicable. 20% par défaut
* Fabriquant du produit
* S’il existe : Grossiste ayant assuré l’intermédiaire de l’achat
* Code-barres

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_PRODUIT | int | Non |
| LIBEL\_PRODUIT | nvarchar(256) | Non |
| DESC\_PRODUIT | nvarchar(1024) | Non |
| ID\_SSFAMILLE\_PRODUIT | int | Non |
| CODE\_BARRE\_PRODUIT | int | Non |
| PRIX\_ACHAT | money | Non |
| TAUX\_TVA | decimal(4, 1) | Non |
| MARQUE\_PRODUIT | nvarchar(256) | Non |
| GROSSISTE\_PRODUIT | nvarchar(256) | Oui |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table CLIENTS

But

Cette table contient tous les clients ayant acheté des produits chez CASTO-MERLIN au cours des 2 derniers mois (Historique flottant). Pour des raisons de volumétrie et de performance, la table est historisée vers une base de données secondaire, configurée pour le stockage en masse en lecture seule (au détriment des performances) pendant 2 ans. Après quoi, les données sont supprimées.

On distingue 5 types de clients (cf. champ « TYPE\_CLIENT ») :

* **A – Anonyme**. Ce sont la majorité des clients de magasins. Des particuliers sans programme de fidélité. L’hôtesse de caisse recueille seulement leur code-postal d’habitation à fins de statistiques.
* **I – Internet**. Ce sont les clients de site Internet sans programme de fidélité. Le site recueille leurs nom et adresse pour livraison et sécurité de la commande (Lutte contre la fraude)
* **N – Nominatif**. Ce sont des clients de type particulier, en magasin ou sur Internet, avec un programme de fidélité. Leur identifiant de carte fidélité est alors renseigné dans le champ « CODE\_CARTE\_FIDEL »
* **P – Professionnel de type artisan**.
* **S – Professionnel de type société**.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_CLIENT | int | Non |
| NOM\_CLIENT | nvarchar(256) | Non |
| TYPE\_CLIENT | char(1) | Non |
| DATE\_NAISSANCE | date | Non |
| DATE\_SOUSCRIPTION | date | Non |
| ID\_VILLE\_CLIENT | int | Non |
| LIBEL\_ADRESSE | nvarchar(256) | Oui |
| CODE\_CARTE\_FIDEL | nvarchar(32) | Oui |
| TAUX\_REMISE | decimal(4, 1) | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table LIEUX

But

Cette table contient tous les lieux de ventes ou de stockage de CASTO-MERLIN depuis la création de l’entreprise. Ce peut être des lieux de ventes (Rayonnages des magasins, site Internet) de stockage (Entrepôts régionaux, partie stocks d’un magasin) ou des plateformes logistiques où les marchandises ne font que transiter.

On distingue 5 types de lieux (cf. champ « TYPE\_LIEU ») :

* **R – Rayons de vente du magasin.**
* **M – Partie stocks du magasin.**
* **I – Site Internet.**
* **S – Partie stocks pour Internet.**
* **E – Entrepôt régional.**
* **P – Plateforme logistique.**

Se reporter à la section « **Base opérationnelle – Vue d’ensemble** » pour la signification de ces termes.

Chaque lieu est dans une ville donnée et une seule. En cas de déménagement, la référence vers la ville sera modifiée. En cas de fermeture, le lieu sera supprimé de cette table dans la base opérationnelle.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_LIEU | int | Non |
| TYPE\_LIEU | char(1) | Non |
| LIBEL\_LIEU | nvarchar(256) | Non |
| ID\_VILLE | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table VILLES

But

Le but de cette table de dimension est de permettre une analyse des ventes suivant l’axe Ville. Il est également possible de définir la hiérarchie suivante : **Region ->Departement->Arrondissement->Commune->Canton->Ville,** à l’effet de mener des analyses de ventes suivant plusieurs niveaux de vue de la table VILLE.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| VILLE\_PK | int | Non |
| CODE\_POSTAL | nvarchar(6) | Non |
| CODE\_COMMUNE | int | Non |
| CODE\_REGION | int | Non |
| CODE\_DEPARTEMENT | int | Non |
| CODE\_ARRONDISEMENT | int | Non |
| CODE\_CANTON | int | Non |
| NOM\_VILLE\_MAJ | nvarchar(256) | Non |
| NOM\_VILLE\_MIN | nvarchar(256) | Non |
| POPULATION | int | Oui |

**Clé primaire** : VILLE\_PK (Clé technique)

Ici, les champs **CODE\_POSTAL, CODE\_COMMUNE, CODE\_REGION, CODE\_DEPARTEMENT, CODE\_ARRONDISEMENT, CODE\_CANTON**, considérés comme les clés étrangères n’ont pas les tables de dimensions associées.

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

/\***BERNARD** : A COMPLETER \*/

## Table STOCKS

But

/\* A COMPLETER SI VALORISATION ULTERIEURE \*/

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_STOCK | int | Non |
| DATE\_RECENSEMENT | datetime | Non |
| OPER\_RECENSEMENT | nvarchar(64) | Non |
| ID\_LIEU | int | Non |
| ID\_PRODUIT | int | Non |
| NBR\_DISPO | int | Non |
| NBR\_DEFECTUEUX | int | Non |
| NBR\_RETOUR\_SAV | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

/\* A COMPLETER SI VALORISATION ULTERIEURE \*/

## Table VENTES

But

La table des ventes est une table d’association entre :

* **Un ticket**, c’est-à-dire de 1 à N articles vendus (Clé étrangère vers la table TICKETS)
* **Un client** (Clé étrangère vers la table CLIENTS)
* **Un lieu**, de type Internet ou magasin (Clé étrangère vers la table LIEUX)

La vente a un montant global, et une somme de TVA, tous deux typés MONEY, plus adapté que DECIMAL au stockage de valeurs financières.

Le champ « Operateur Ventes » stocke, pour information, le matricule de l’employée ayant réalisé la vente (0 si site Internet)

Exception du SI opérationnel de ventes de CASTO-MERLIN : Il ne comprend que le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »)

Il n’y a pas de champs pour le nom d’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date car cette table n’est pas mise à jour une fois la vente effectuée. En cas d’annulation de la vente, la ligne n’est jamais insérée, et en cas de remboursement, il est créé une vente de montant opposé à la vente à rembourser.

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_VENTE | int | Non |
| OPER\_VENTE | nvarchar(64) | Non |
| ID\_TICKET | int | Non |
| ID\_CLIENT | int | Non |
| ID\_LIEU | int | Non |
| MONTANT\_HT\_VENTE | money | Non |
| MONTANT\_TVA\_VENTE | money | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table TICKETS

But

Comme en magasin, le ticket est un ensemble de 1 à N produits vendus. On ne précise ni le client ni le lieu (Ces informations sont reprises dans la table VENTES)

On en relie pas directement les produits, mais les couples « Prix – Produits », car un même produit peut avoir un prix diffèrent en fonction des magasins et de leur politique propre de prix.

Si dans un même ticket, le client achète plusieurs fois le même produits, alors le champ QUANTITE sera diffèrent de 1.

Exception du SI opérationnel de ventes de CASTO-MERLIN : Il ne comprend que le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »)

Il n’y a pas de champs pour le nom d’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date car cette table n’est pas mise à jour une fois la vente effectuée. En cas d’annulation de la vente, la ligne n’est jamais insérée, et en cas de remboursement, il est créé une vente de montant opposé à la vente à rembourser.

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_TICKET | int | Non |
| ID\_PRIXPRODUIT | int | Non |
| QUANTITE | int | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |

Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

## Table PRIXPRODUITS

But

La tale PRIXPRODUITS fixe le prix d’un produit dans un lieu donné. En effet, un même produit peut avoir un prix diffèrent en fonction des magasins et de leur politique propre de prix.

Le champ « Operateur Prix Produit » stocke, pour information, le matricule de l’employée ayant fixé le prix du produit sur le lieu de vente donné.

On peut noter une légère entorse à la normalisation du modèle relationnel :

* La table VENTES relie (Entre autre) un lieu à un ticket
* La table TICKETS relie un ticket à un Prix-produit
* La table PRIXPRODUIT relie un produit à un prix dans un lieu donné

On voit donc une redondance de l’information « lieu » entre la table des VENTES et celle des PRIXPRODUITS. Cette information a été dupliquée pour améliorer la lisibilité et l’exploitation de la table des VENTES.

Comme la quasi-totalité des tables du SI opérationnel de ventes de CASTO-MERLIN, il comprend le nom de l’utilisateur ayant fait l’INSERT de cette ligne, et la date (Resp. les champs « OPER\_CREAT » et « DATE\_CREAT »), ainsi que le nom de l’utilisateur ayant fait le dernier UPDATE de cette ligne, et la date (Resp. les champs « OPER\_MODIF » et « DATE\_MODIF »)

L’opérateur peut être le nom d’une personne (« Jean DUPOND ») ou d’un SI externe (« IBM Point Of Sales System »)

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| ID\_PRIXPRODUIT | int | Non |
| OPER\_PRIXPRODUIT | nvarchar(64) | Non |
| ID\_LIEU | int | Non |
| ID\_PRODUIT | int | Non |
| MONTANT\_HT\_PRODUIT | money | Non |
| MONTANT\_TVA\_PRODUIT | money | Non |
| MONTANT\_MARGE\_PRODUIT | money | Non |
| DATE\_CREAT | datetime | Non |
| OPER\_CREAT | nvarchar(64) | Non |
| DATE\_MODIF | datetime | Oui |
| OPER\_MODIF | nvarchar(64) | Oui |

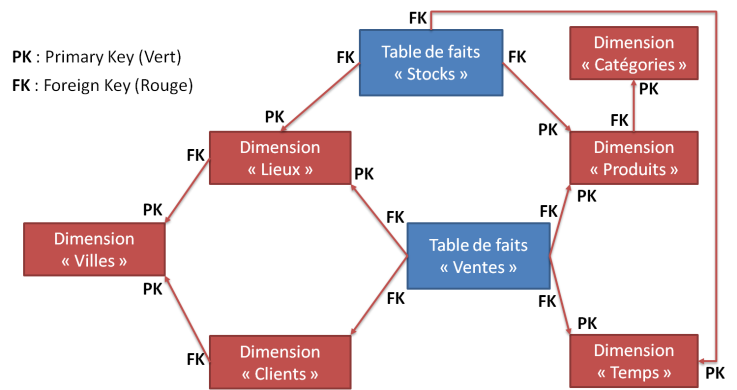
Remplissage

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

# Entrepôt de données

## Vue d’ensemble

Dans la base dédiée « **DataWarehouseODE** », nous créons un schéma dédié « **ODE\_DATAWAREHOUSE** », qui contient les tables suivantes :



## Table de dimension CATEGORIES

But

La table CATEGORIES est une table de dimension qui permet de choisir une catégorie de produit en fonction des univers, famille ou sous-famille des produits. Ces informations servent à aiguiller ou cibler les recherches sur les catégories de produits.

Exemple :  Rangement et dressing

 > Accessoires de rangement

 > Panier, malle et boite de rangement

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| CATEGORIE\_PK | int | Non |
| LIBEL\_UNIVERS | nvarchar(256) | Oui |
| ID\_UNIVERS | int | Oui |
| LIBEL\_RAYON | nvarchar(256) | Oui |
| ID\_RAYON | int | Oui |
| LIBEL\_FAMILLE | nvarchar(256) | Oui |
| ID\_FAMILLE | int | Oui |
| LIBEL\_SSFAMILLE | nvarchar(256) | Oui |
| ID\_SSFAMILLE | int | Oui |

**Clé primaire** : CATEGORIE\_PK (Clé technique)

**Clé étrangère** : ∅ (C’est une table de dimension en fin de branche du flocon)

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 : ETL \*/

Remplissage pour les tests

**Brève explication des principaux champs**

* Le libellé « Univers » qui servira à déterminer dans quel univers se retrouve une catégorie. Par exemple : Univers « Rangement et dressing »
* Le libellé « Rayon » qui sert à déterminer le rayon où retrouver la catégorie de produit.
* Les libellés « Famille » et « Sous-famille » qui serviront à cibler la recherche. Par exemple : Accessoires de rangement > Panier, malle et boite de rangement

Ce découpage permettre de déterminer rapidement les catégories des produits donc d’effectuer un ciblage rapide et efficace.

## Table de dimension PRODUITS

But

La table PRODUITS est une table de dimension dont le but est de pouvoir analyser les ventes cet axe (ventes par marque, par fournisseur, par produit etc.) mais également de calculer certains indicateur de mesures grâce aux PRIX\_ACHAT et à au TAUX\_TVA notamment.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| PRODUIT\_PK | int | Non |
| CATEGORIE\_FK | int | Non |
| LIBEL\_PRODUIT | nvarchar(256) | Non |
| PRIX\_ACHAT | money | Non |
| TAUX\_TVA | decimal(4, 1) | Non |
| MARQUE\_PRODUIT | nvarchar(256) | Non |
| GROSSISTE\_PRODUIT | nvarchar(256) | Oui |

**Clé primaire** : PRODUIT\_PK (Clé technique)

**Clé étrangère** : CATEGORIE\_FK → CATEGORIES.CATEGORIE\_PK

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

La table PRODUITS a également été dénormalisée en ajoutant une table CATEGORIE\_PRODUIT (cf. ci-après) toujours dans la même optique que pour la table VILLE : gagner en performance de mise à jour. En effet, la table PRODUITS sera sans doute mise à jour de manière très régulière alors que les CATEGORIE\_PRODUIT subiront sans doute moins de changements.

La table PRODUITS contient donc une clé primaire et une clé étrangère correspondant à la clé primaire de la table CATEGORIE\_PRODUIT ainsi que les champs :

* Libellé produit, par exemple : « Rouleau antigoutte L180 »
* Le prix d'achat du produit qui servira à calculer la mesure Marge.
* Le taux de TVA du produit qui servira également pour les indicateurs financiers.
* La marque du produit qui permettra par exemple d'effectuer des comparaisons entre marque sur un même type de produit et de réajuster les approvisionnements ensuite.
* Le fournisseur du produit qui pourra également permettre d'affiner la stratégie d'achat en regroupant les produits à forte demande auprès d'un seul fournisseur en négociant les prix par exemple.

## Table de dimension TEMPS

But

La table TEMPS est une table de dimension utilisée pour quantifier la dimension d’analyse « Temps » des faits de ventes et de stocks.

Cette dimension est une quasi-obligatoire dans les SI décisionnels d’entreprise.

Nous la limiterons à la granularité « Jour », et non heure. En effet, les métiers du Marketing et du Commercial ne requièrent pas une telle finesse.

On définit 4 « groupes » de temps :

* Année
* Trimestre
* Mois
* Semaine

Chaque groupe est défini simultanément par :

* Un libellé, pour l’affichage à l’utilisateur
* Une date, qui correspond au Jour-Mois-Année du début de la période.

De plus, on pré-enregistre les référence du jour de chacun de ces groupes :

Par exemple, JOUR\_DU\_MOIS indique le numéro de jour dans le mois, entre 1 et 31 pour mai, entre 1 et 30 pour juin, etc…

Structure

|  |  |  |
| --- | --- | --- |
| **om colonne** | **Type** | **Nullable ?** |
| TEMPS\_PK | DATETIME | N |
| DATE\_NOM | NVARCHAR(50) | O |
| ANNEE | DATETIME | O |
| ANNEE\_NOM | NVARCHAR(50) | O |
| TRIMESTRE | DATETIME | O |
| TRIMESTRE\_NOM | NVARCHAR(50) | O |
| MOIS | DATETIME | O |
| MOIS\_NOM | NVARCHAR(50) | O |
| SEMAINE | DATETIME | O |
| SEMAINE\_NOM | NVARCHAR(50) | O |
| JOUR\_DE\_ANNEE | INT | O |
| JOUR\_DE\_ANNEE\_NOM | NVARCHAR(50) | O |
| JOUR\_DU\_TRIMESTRE | INT | O |
| JOUR\_DU\_TRIMESTRE\_NOM | NVARCHAR(50) | O |
| JOUR\_DU\_MOIS | INT | O |
| JOUR\_DU\_MOIS\_NOM | NVARCHAR(50) | O |
| JOUR\_DE\_SEMAINE | INT | O |
| JOUR\_DE\_SEMAINE\_NOM | NVARCHAR(50) | O |
| SEMAINE\_DE\_ANNEE | INT | O |
| SEMAINE\_DE\_ANNEE\_NOM | NVARCHAR(50) | O |
| MOIS\_DE\_ANNEE | INT | O |
| MOIS\_DE\_ANNEE\_NOM | NVARCHAR(50) | O |
| MOIS\_DU\_TRIMESTRE | INT | O |
| MOIS\_DU\_TRIMESTRE\_NOM | NVARCHAR(50) | O |
| TRIMESTRE\_DE\_ANNEE | INT | O |
| TRIMESTRE\_DE\_ANNEE\_NOM | NVARCHAR(50) | O |

**Clé primaire** : TEMPS\_PK (Clé technique)

Contrairement aux autres tables de dimensions, la clé primaire technique n’est pas auto-incrémentée (Mot-clé IDENTITY dans les scripts de création de table) En effet, pour des raisons d’optimisation de performances et de clarté, les clés primaires sont au format DATETIME « AAAAMMDD ». Par exemple : « 31/12/2015 00 :00 :00 » pour 31 décembre 2015.

**Clé étrangère** : ∅

Il n’y a pas de clé étrangère car c’est une table de dimension en fin de branche du flocon. Elle ne prend donc plus de référence dans d’autres tables de dimensions.

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

Afin de prendre large, elle a été remplie du 01 janvier 2009 au 31 décembre 2018 (Soit 3 652 jours)

Les outils graphiques de SQL Server 2014 proposent des assistants de création de cette dimension « standard », qui a été utilisé pour générer la structure et le contenu de cette table.

## Table de dimension LIEUX

But

La table LIEUX est une table de dimension qui permettra d'analyser les ventes selon les différents lieux : type de lieu, ville d'implantation etc.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| LIEU\_PK | int | Non |
| VILLE\_FK | int | Non |
| TYPE\_LIEU | char(1) | Non |
| LIBEL\_LIEU | nvarchar(256) | Non |
| DATE\_OUVERTURE | date | Non |
| DATE\_FERMETURE | date | Non |
| SURFACE\_M2 | numeric(6, 1) | Oui |

**Clé primaire** : LIEU\_PK (Clé technique)

**Clé étrangère** : VILLE\_FK → VILLES.VILLE\_PK

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

Nous avons choisis de relier la table LIEUX à la table VILLE par une clé étrangère représentant l'identifiant unique de la Ville. A noter que la table VILLE est également reliée à la table CLIENT puisque ces derniers habitent dans des villes : cette solution est cohérente comme l'indique Thomas Gauchet dans son livre SQL Server 2014 : Implémentation d'une solution de Business Intelligence. L'autre solution aurait été de créer deux tables : VILLES\_CLIENTS et VILLES\_LIEUX ou encore d'intégrer les informations de ces tables directement dans les tables LIEUX et CLIENTS. Notre choix s'est basé sur le critère de la limitation du nombre de tables (une seule table VILLE dans notre solution) tout en dénormalisant les informations des villes pour gagner en performance puisque celles-ci ne seront sans doute pas mise à jour très régulièrement en comparaison de la table CLIENTS.

En plus d'une clé primaire et d'une clé étrangère (vers la table VILLE), la table LIEUX contient les champs :

* Type de lieu, permettant d'identifier un lieu parmi Magasin, Entrepôt, Stock Magasin, Stock site Internet, Site Internet et Plateforme logistique à l'aide d'un caractère.
* Le libellé du lieu, par exemple : « Magasin de Talence ZAC (33) ».
* Les dates d'ouverture et de fermeture du magasin du magasin : pour permettre de comparer les performances des magasins en fonction de leurs "ancienneté".
* La surface du magasin qui nous permettra de matérialiser la performance des magasins en terme de CA/m² ou de Quantité vendues/m². Cet indicateur permettra de mesurer non seulement la performance d'un magasin (et éventuellement d'adapter les meilleurs techniques d'organisation pour les autres magasins) mais également de ramener le Chiffre d'Affaires et les quantité vendues à des ratios comparables entre établissements : c'est à dire pouvoir comparer un grand magasin de 1000m² de la banlieue bordelaise avec un magasin de 200m² du centre de la Creuse.

## Table de dimension CLIENTS

But

La table CLIENTS est une table de dimension qui permet d’analyser les ventes et les stocks en fonction des critères du client. Cette table contient tous les clients ayant acheté des produits chez CASTO-MERLIN au cours des 2 derniers mois (Historique flottant). Elle permet des analyses en fonction du client sur son type(Anonyme, Internaute, Nominatif, Artisan ou encore une société), son taux de remise ou d’autre donnée.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| CLIENT\_PK | int | Non |
| VILLE\_FK | int | Non |
| TAUX\_REMISE | decimal(6, 2) | Non |
| TYPE\_CLIENT | char(1) | Non |
| NOM\_CLIENT | nvarchar(256) | Non |
| DATE\_NAISSANCE | date | Non |
| DATE\_SOUSCRIPTION | date | Non |
| CODE\_FIDELITE | nvarchar(32) | Oui |

**Clé primaire** : CLIENT\_PK (Clé technique)

**Clé étrangère** : VILLE\_FK → VILLES.VILLE\_PK

La table Clients possède deux clefs, une clef primaire dite Technique et une clef étrangère permettant le lien vers la table Ville afin de récupérer les informations liées à la ville du souscripteur.

Les champs supplémentaires suivants sont ajoutés afin de permettre certain rapprochement :

* Taux de remise qui correspond au pourcentage accordé au client.
* Type de client donnant une information sur le statut du client. Les clients ont été regroupés en 5 catégories :
  + **A – Anonyme**. Ce sont la majorité des clients de magasins. Des particuliers sans programme de fidélité. L’hôtesse de caisse recueille seulement leur code-postal d’habitation à fins de statistiques.
  + **I – Internet.** Ce sont les clients de site Internet sans programme de fidélité. Le site recueille leurs nom et adresse pour livraison et sécurité de la commande (Lutte contre la fraude)
  + **N – Nominatif**. Ce sont des clients de type particulier, en magasin ou sur Internet, avec un programme de fidélité. Leur identifiant de carte fidélité est alors renseigné dans le champ « CODE\_CARTE\_FIDEL ».
  + **P – Professionnel de type artisan**.
  + **S – Professionnel de type société.**
* Nom du client qui est présent uniquement à titre informatif car difficilement analysable.
* Date de naissance qui correspond à la date de naissance du client si elle est connue (par défaut, elle sera à 01.01.0001).
* Date de souscription qui correspond à la date de mise en base du nouveau client.
* Code fidélité qui correspond à l’identifiant de la carte de fidélité du client.

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

Le remplissage de la table pour les tests a été effectué par script T-SQL. Ce script permet de déterminer le nombre de postes souhaités en fin de chargement afin de pouvoir paramétrer un éventuel échantillonnage.

La répartition des clients par type est la suivante :

* Anonyme : 40%
* Internaute : 20%
* Nominatif : 20%
* Professionnel : 10%
* Société : 10%

Le reste des champs ont été choisis de la façon suivante :

* Ville : choix aléatoire dans la table Ville
* Taux de remise : choix aléatoire entre 0 et 100 et forcé à 0 si supérieur à 50%.
* Nom du client :
  + Le nom du client sera généré avec une longueur (entre 0 et 100) et un contenu (alternance consonne et voyelle) aléatoire. La longueur a été choisie de façon aléatoire afin de posséder une table avec de vrai varchar et non des champs « nom » de longueur fixe. Un client anonyme aura un nom « Client anonyme ».
  + Pour les internautes, on suffixera le nom par « INT » suivi du numéro d’enregistrement.
  + Pour les clients nominatifs, on suffixera le nom par « BOB » suivi du numéro d’enregistrement.
  + Pour les sociétés, le nom sera suffixé par « SARL ».
  + Pour les professionnels, le nom sera suffixé par « PRO » suivi du numéro d’enregistrement.
* Date de naissance : choisie de façon aléatoire sur les 80 dernières années pour les internautes, les nominatifs et les artisans. Pour les autres types de client, le champ est alimenté par sa valeur par défaut « 01.01.0001 ».
* Date de souscription : choisie aléatoirement sur les 20 dernières années.
* Code fidélité : Il est généré avec une longueur (entre 0 et 32) et un contenu (alternance consonne et voyelle) aléatoire si le taux de remise est supérieur à 0. Sinon il sera à blanc.

## Table de dimension VILLES

But

Le but de cette table de dimension est de permettre une analyse des ventes suivant l’axe Ville. Il est également possible de définir la hiérarchie suivante : **Region ->Departement->Arrondissement->Commune->Canton->Ville,** à l’effet de mener des analyses de ventes suivant plusieurs niveaux de vue de la table VILLE.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| VILLE\_PK | int | Non |
| CODE\_POSTAL | nvarchar(6) | Non |
| CODE\_COMMUNE | int | Non |
| CODE\_REGION | int | Non |
| CODE\_DEPARTEMENT | int | Non |
| CODE\_ARRONDISEMENT | int | Non |
| CODE\_CANTON | int | Non |
| NOM\_VILLE\_MAJ | nvarchar(256) | Non |
| NOM\_VILLE\_MIN | nvarchar(256) | Non |
| POPULATION | int | Oui |

**Clé primaire** : VILLE\_PK (Clé technique)

**Clé étrangère** : ∅ (C’est une table de dimension en fin de branche du flocon)

Ici, les champs **CODE\_POSTAL, CODE\_COMMUNE, CODE\_REGION, CODE\_DEPARTEMENT, CODE\_ARRONDISEMENT, CODE\_CANTON**, considérés comme les clés étrangères n’ont pas les tables de dimensions associées. Nous avons fait le choix de ne pas développer le schéma en flocon pour la table VILLE.

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

/\***BERNARD** : A COMPLETER \*/

## Table de faits VENTES

But

Cette table de faits contient toutes les ventes depuis la mise en service du SI décisionnel, le 01 janvier 2010.

Chaque ligne enregistre la vente d’un produit (Ou de plusieurs identiques, cf. champs quantité) à un client, sur un magasin ou sur Internet, et à une date précise.

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| DATE\_VENTE\_FK | int | Non |
| PRODUIT\_FK | int | Non |
| CLIENT\_FK | int | Non |
| LIEU\_FK | int | Non |
| MONTANT\_HT\_VENTE | money | Non |
| MONTANT\_TVA\_VENTE | money | Non |
| MARGE\_BRUTE | money | Non |
| UNITES\_VENDUES | int | Non |
| NUM\_TICKET | nvarchar(256) | Non |

**Clé primaire** : ∅

La table de Ventes étant une table de faits, elle ne contient aucune clé primaire, qui n’a aucune utilité puisque les faits ne sont pas exploités individuellement, mais collectivement par le biais de regroupements et de statistiques.

**Clés étrangères** :

* DATE\_VENTE\_FK → TEMPS.TEMPS\_PK
* PRODUIT\_FK → PRODUITS.PRODUIT\_PK
* CLIENT\_FK → CLIENTS.CLIENT\_PK
* LIEU\_FK → LIEUX.LIEU\_PK

Fonctionnellement, chaque Vente lie un client, un lieu (Le magasin ou le site Internet) et un produit, à une date donnée. Techniquement, la table de faits Ventes utilise des clés étrangère vers chacune de ces dimensions.

Remplissage via l’ETL

/\* A COMPLETER LORS DU LOT 3 \*/

Remplissage pour les tests

Pour chaque année entre 2010 et 2015, on considère un volume de ventes stable :

* 2.2 Millions de ventes annuelles sur Internet
* 9.6 Millions de ventes annuelles pour l’ensemble des magasins physiques

Concernant le champ « Marge brute » :

* Entre 24.3 et 64.3 % de marge sur Internet - Moyenne à 44.3 %
* Entre 14.8 et 54.8 % de marge en Magasin - Moyenne à 34.8 %

Qui correspond à de « bons » chiffres dans la grande distribution de Bricolage.

Concernant le nombre moyens d’articles par panier :

* Entre 1 et 9 articles sur Internet - Moyenne de 5
* Entre 1 et 35 articles en Magasin - Moyenne de 18

Dans ces deux cas : On commence par choisir un client et un ticket de caisse, puis on ajoute un nombre aléatoire d’articles pour rentrer dans cette fourchette de nombre d’articles. Le nombre d’articles identiques ajoutés dans le panier d’un client est stocké dans le champ « Unités vendues ». En effet, il est fréquent qu’au cours d’un acte d’achat, le client prenne plusieurs fois exactement le même produit (Exemple : 5 planches, 20 paquets de clous, 2 tournevis…) Afin d’économiser de la place, nous regroupons ces ventes d’articles identiques à un même client dans une seule ligne de table de faits Ventes.

Le champ « Montant HT » va contenir le montant total HT de la vente, c’est-à-dire le prix d’achat du produit (Cf. table de dimension « Produits ») plus la marge brute, multiplié par le nombre d’articles identiques vendus (Cf. champ « Unités vendues »)

Le champ « Montant TVA » représente la TVA de cette même vente. Le taux appliqué sur ce produit est précisé dans la table de dimension « Produits ».

Enfin, le champ « Numero de Ticket » est dit « dégénéré » : Il s’agit d’une clé primaire du SI opérationnel des ventes. Ce champs n’est utile que pour remonter à la source de la vente (« Pour information ») et ne sera jamais valorisé dans le SI décisionnel.

## Table de faits STOCKS

But

/\* A COMPLETER SI VALORISATION ULTERIEURE \*/

Structure

|  |  |  |
| --- | --- | --- |
| **Nom colonne** | **Type** | **Nullable ?** |
| DATE\_INVENTAIRE\_FK | int | Non |
| PRODUIT\_FK | int | Non |
| LIEU\_FK | int | Non |
| NBR\_DISPO | int | Non |
| NBR\_DEFECTUEUX | int | Non |
| ID\_INVENTAIRE | int | Non |

**Clé primaire** : ∅ (C’est une table de faits)

**Clés étrangères** :

* DATE\_INVENTAIRE\_FK → TEMPS.TEMPS\_PK
* PRODUIT\_FK → PRODUITS.PRODUIT\_PK
* LIEU\_FK → LIEUX.LIEU\_PK

Remplissage via l’ETL

/\* A COMPLETER SI VALORISATION ULTERIEURE \*/

Remplissage pour les tests

/\* A COMPLETER SI VALORISATION ULTERIEURE \*/