

16/12/2018

Business Intelligence System & Data Warehouse

BD51 – Automne 2018



Réalisé par:

KENAAN Assaad
Génie informatique
GI05
ALWAN Marwan
Génie informatique
GI05

Encadré par:

M. FISCHER Christian
Responsable des unités de valeur :
BD40, BD50, BD51, LO51

Table des matières

I.	Introduction	2
II.	Objectif et Scénario.....	3
III.	Partie 1 : Fonctions d’ETL.....	4
1)	Qualité des données	4
2)	Packages SSIS	4
3)	Détail de l’objectif des packages SSIS	5
a)	P1_TRANSFER_DATA.....	5
b)	P2_Audit_INCR_TRANSFER_DATA	5
c)	P3_TRACABILITY_TRANSFER_DATA	6
d)	P4_TRANSFER_AGGREGATE_TABLE.....	6

I. Introduction

Dans le cadre de notre formation à l'**Université de Technologie de Belfort Montbéliard (UTBM)**, et dans le cadre de l'**UV BD51 (Business Intelligence & Data Warehouse)**, nous sommes amenés à concevoir et développer un système décisionnel pour la gestion des ventes par magasin pour la base de données **EMODE**.

Pour cela, le système est réalisé sur une machine virtuelle ayant comme système d'exploitation Windows 10, 64 bits, version anglaise. Les systèmes de gestion de base de données (SGBD) utilisés sont Oracle 12C et SQL Server 2016.

L'objectif de ce rapport est de mettre l'accent et expliquer en détail les différentes étapes qui nous ont emmenées à la réalisation du projet, en partant de la vérification de la qualité des données, passant par la mise en œuvre des fonctions d'ETL (Extract Transform Load) et la synchronisation des données entre les deux sources, ensuite en passant par l'optimisation du Data Warehouse et enfin par la mise en place des rapports afin assurer l'interrogation des bases de données selon les requêtes SQL préparées lors de l'élaboration du modèle.

II. Objectif et Scénario

L'objectif final attendu de ce projet est d'obtenir un système qui permet à un utilisateur d'avoir une vision claire et précise sur l'état des ventes par magasin de la base de données EMODE. Pour cela, des rapports seront faites et qui priorisent les caractéristiques désirées par l'utilisateur.

Pour atteindre cet objectif, nous avons décomposé les tâches en plusieurs étapes. D'abord, nous devons effectuer un transfert de la totalité des données de la base Oracle vers une base SQL Server. Pour ce faire, nous devons se servir des fonctionnalités d'ETL, et précisément on a créé des packages SSIS qui organise au mieux cette tâche.

Ensuite, nous avons optimisé les données transférées afin de diminuer au minimum le temps de calcul et d'améliorer la performance de notre application.

Puis, nous avons mis en place par le biais d'un projet Analysis Services, un cube OLAP afin de naviguer au sein des données et d'avoir un aperçu de la qualité de celles-ci.

Enfin, nous avons mis en place différents types de rapports, le tout à l'aide plusieurs outils comme le Reporting Services et Web Intelligence.

Comme c'est indiqué dans le sujet, nous avons travailler juste avec les tables suivantes :

- ✓ ARTICLE_COLOR_LOOKUP
- ✓ ARTICLE_LOOKUP
- ✓ OUTLET_LOOKUP
- ✓ CALENDAR_YEAR_LOOKUP
- ✓ SHOP_FACTS

Ces tables seront donc utilisées dans chaque rapport et interviennent aussi dans l'univers BO.

III. Partie 1 : Fonctions d'ETL

1) Qualité des données

Durant le transfert de données, il se peut arriver que certaines données ne se soient pas conformes, c.à.d. qu'elles ne respectent pas les contraintes d'unicités (Clés primaires) ainsi que les contraintes de clés étrangères dans les tables références. Afin d'éviter tout mal transfert, on du rédiger des requêtes SQL coté source de données, qui détectent toute donnée non conforme.

Une fois la ou les données sont détectées, elles seront transmises vers des tables de rejet créer au préalable dans la destination (SQL Server). Pour simplifier les choses, ces tables ont le même nom de celles de la source de données mais avec un préfixe '_TRASH'.

Cette étape est primordiale car elle nous permette d'avoir une traçabilité sur toutes les données que ça soit conformes ou non conformes, et d'éviter tout perte de données, ce qui n'est pas du tout désirées dans notre cas. Grace à cette étape ou pourra rectifier ces données non transférées.

On se sert de ces requêtes lors du transfert des données et précisément dans les packages SSIS.

(FIGURE Tables de rejet sur EMODE de SQL SERVER)

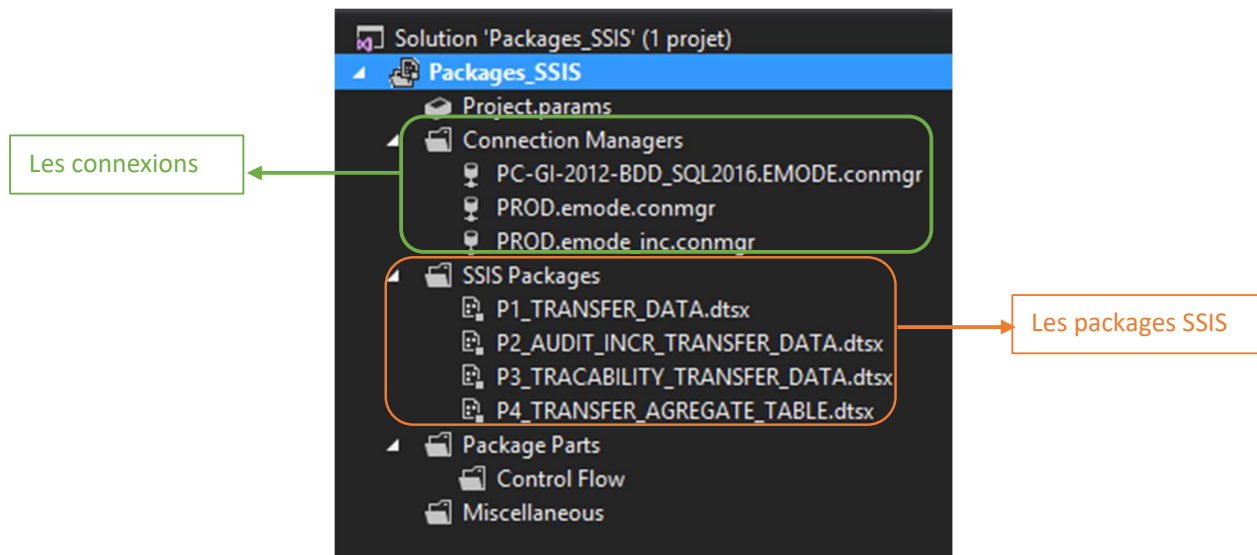
2) Packages SSIS

Les packages SSIS sont créés pour transférer les données depuis Oracle vers SQL Server. On a créé 4 packages avec Business Intelligence Development Studio dont chacun exécute un traitement différent pour résoudre une problématique spécifique.

Voici ci-dessous un tableau qui récapitule l'objectif de chaque package SSIS.

Nom du package	Objectif
P1_Transfer_DATA	Permet de transférer toutes les données depuis Oracle (EMODE) vers SQL Server, tout en prenant en compte les données non conformes.
P2_AUDIT_INCR_TRANSFER_DATA	Permet de transférer les nouvelles données (ajoutées, modifiées ou supprimées) d'Oracle, en consultant les tables Audit créées à la main.
P3_TRACABILITY_DATA	Permet d'exécuter la même tâche que le deuxième package mais cette fois en ajoutant des informations permettant d'avoir un système de traçabilité détaillé.
P4_TRANSFER_AGGREGATE_TABLE	Permet de supprimer les données des tables agrégées pour qu'ensuite mettre à jour les données à nouveau

Voici ci-dessous un aperçu de l'architecture de notre projet montrant les 4 packages créés, ainsi que les connexions aux bases de données créées.



3) Détail de l'objectif des packages SSIS

a) P1_TRANSFER_DATA

L'objectif du premier package se divise en plusieurs tâches :

- ✓ Désactiver les contraintes des clé étrangères des tables destinations.
- ✓ Vider les tables de destinations dans SQL Server
- ✓ Réactiver les contraintes
- ✓ Appel aux requêtes SQL qui vérifient si les données sont conformes ou non.
- ✓ Transférer les données conformes vers les tables qui leurs correspondent
- ✓ Transférer les données non conformes vers les tables TRASH (créées à la main) qui correspond chaque table

PHOTO d'un data flow du package 1

b) P2_Audit_INCR_TRANSFER_DATA

L'objectif du deuxième package vise les données récemment ajoutées, modifiées ou supprimées. Pour ce faire, il fallait crée des tables Audit pour chaque table source Oracle. Ces tables Audit permettent d'enregistrer n'importe quel type de modification dans les tables sources. Afin de les alimenter par les données, il faut auparavant crée des triggers. Ces triggers permettent de fournir les informations nécessaires sur les données (Leurs ID) et l'action qui a été faite (insertion, mise à jour, suppression) afin de les détecter via les packages SSIS.

Parmi les informations que fournies les tables Audit sont : l'action, le transfert (O comme OK et N comme Non transférer) et l'ID de la ligne en question, etc.

Le deuxième package va consulter chaque table Audit, et vérifie d'abord s'il la ligne est transféré ou non, ensuite récupère l'action s'il s'agit du deuxième cas.

Ensuite il va vérifier la conformité des données, si tout est bon, le transfert aura lieu et sinon, il va générer une erreur qui sera signalé dans une table nommée `TRACE_TABLE_DETAIL`, qui informe par la suite l'utilisateur du non transfert de la donnée et de la cause possible. Il s'agit donc d'une synchronisation des données uniquement conformes et qui respecte les contraintes.

c) `P3_TRACABILITY_TRANSFER_DATA`

L'objectif de ce troisième package est similaire que celui du deuxième mais une tâche supplémentaire sera faite, et qui consiste à récupérer plus d'informations sur le **package_id**, **machine_name**, **transfert_user_name**, **l'execution_user_name** et **l'execution_instance_guid**.

Afin d'avoir un bon système de traçabilité, ce package va fournir les informations nécessaires sur le poste de travail par exemple dans le cas où le transfert est effectué ou non.

Photo des composants ajouter P3

d) `P4_TRANSFER_AGGREGATE_TABLE`

L'objectif du dernier package est d'être lancer à chaque exécution de chaque package. Son objectif est de vider les tables d'agrégats et insérer de nouveau les données.

Photo du package

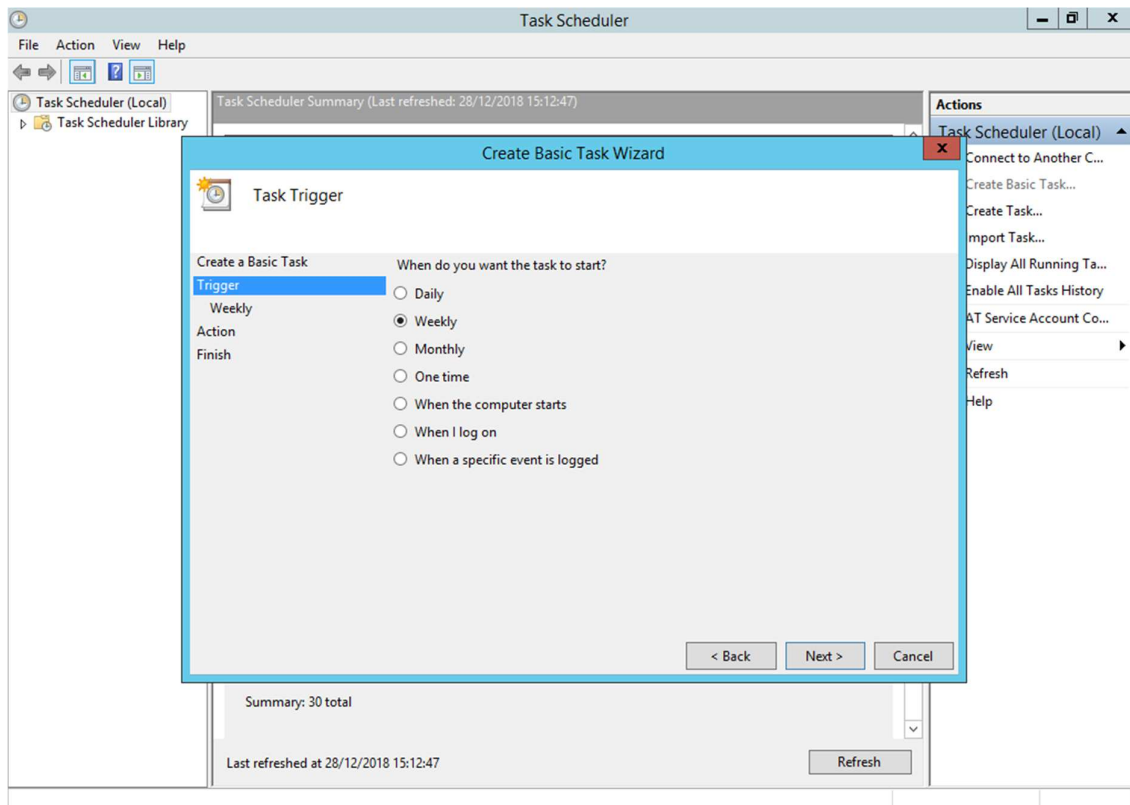
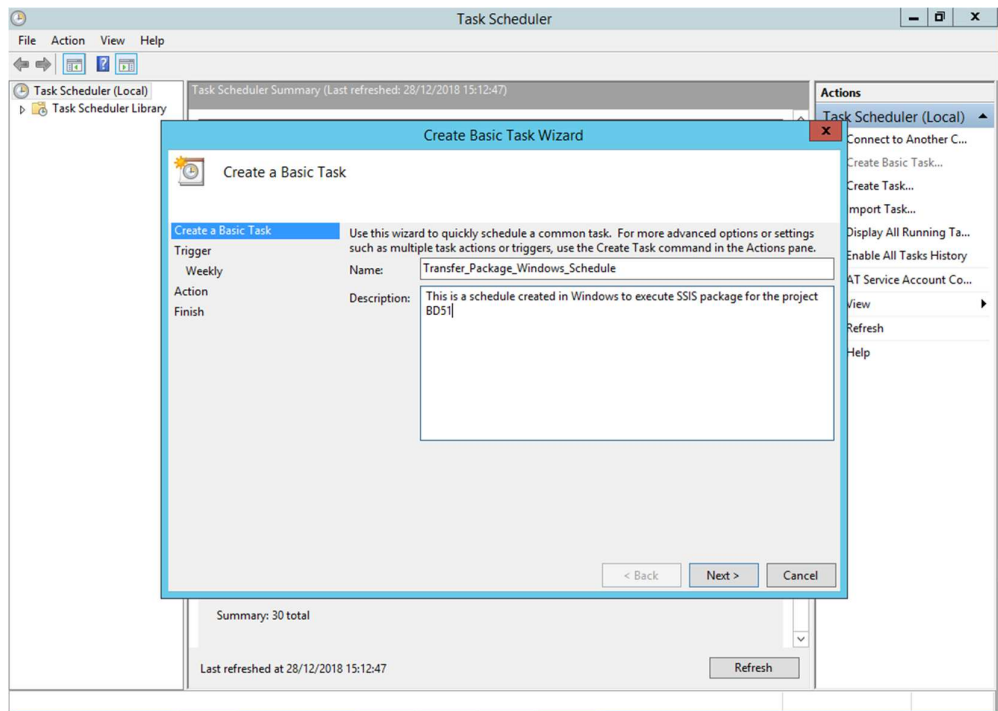
4) Automatisation

Afin d'automatiser les tâches des 4 packages SSIS, il existe pour cela trois méthodes qu'on peut utiliser. La première consiste à exécuter immédiatement le package en appuyant tout simplement sur **démarrer** dans Business Intelligence Development.


PHOTO

Une deuxième méthode appelé **Schedule task** gérer par le système d'exploitation Windows, permet de configurer les tâches d'exécution des packages par ordre de préférence, en précisant les détails sur les dates d'exécution.

Voici ci-dessous la procédure permettant la création de cette tâche.



Create Basic Task Wizard

 **Weekly**

Create a Basic Task

Trigger

Start: 28/12/2018 00:00:00 ☐ Synchronize across time zones


Recur every: 1 weeks on:

☒ Sunday ☐ Monday ☐ Tuesday ☐ Wednesday

☐ Thursday ☐ Friday ☐ Saturday

< Back Next > Cancel

Create Basic Task Wizard

 **Summary**

Create a Basic Task

Trigger

Weekly

Action

Start a Program

Finish

Name: Transfer_Package_Windows_Schedule

Description: This is a schedule created in Windows to execute SSIS package for the project BD51

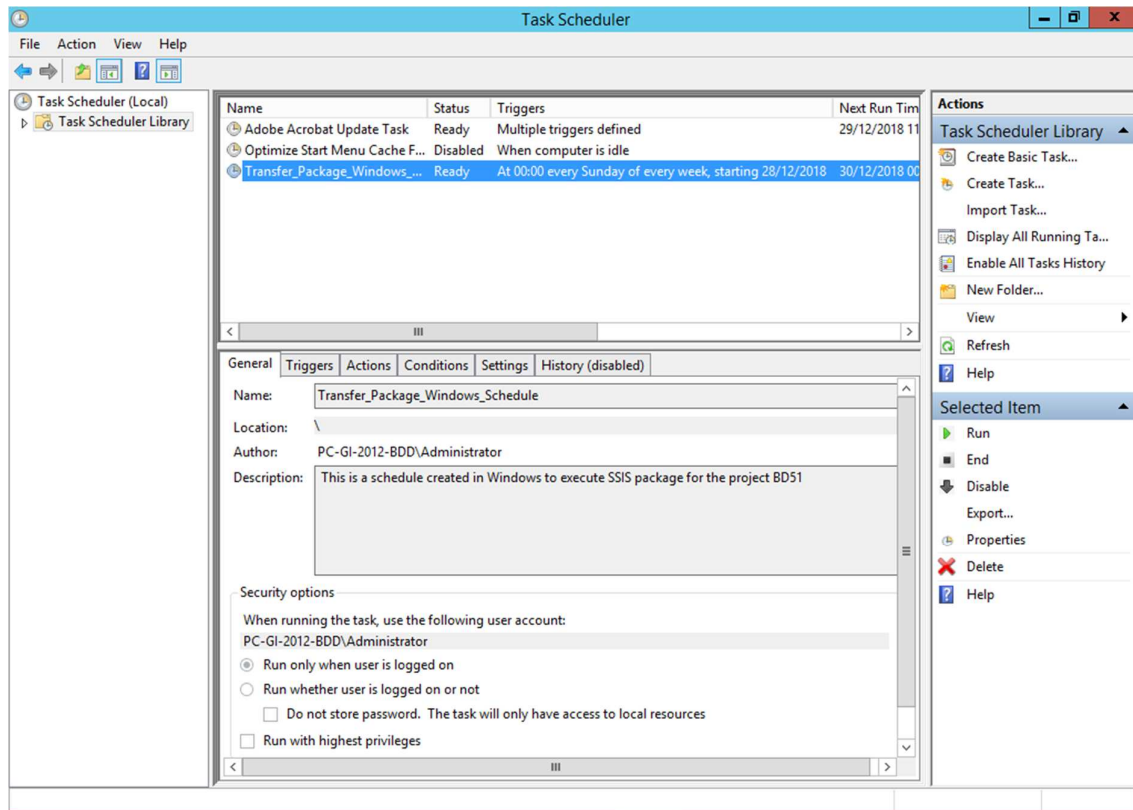
Trigger: Weekly; At 00:00 every Sunday of every week, starting 28/12/2018

Action: Start a program; "E:\Projet_A18\BD51_Projet_Kenaar_Alwan_A18\Partie 1 - ET

☐ Open the Properties dialog for this task when I click Finish

When you click Finish, the new task will be created and added to your Windows schedule.

< Back Finish Cancel



La troisième méthode (**Schedule task SQL Server Agent**) est similaire à la précédente, mais cette fois elle est faite avec SQL Server Agent, qui nous permet de créer une seule tâche pour tous les packages à la fois. Tout en créant un step ou un pas pour chaque package et toujours dans l'ordre de préférence.

Voici ci-dessous la procédure permettant la création de cette tâche.

Job Properties - Package_SSIS_Execute_Job

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

Script Help

Name: Package_SSIS_Execute_Job

Owner: PC-GI-2012-BDD\Administrator

Category: [Uncategorized (Local)]

Description: This is a schedule created within SQL Agent in SQL Server Management Studio to execute SSIS packages for the project BD51

Connection: Server: PC-GI-2012-BDD\SQL2016, Connection: PC-GI-2012-BDD\Administrator, View connection properties

Progress: Ready

Enabled: ☒ Enabled

Source:

Created: 01/01/2019 16:59:05

Last modified: 01/01/2019 17:03:02

Last executed: 01/01/2019 17:04:00

View Job History

OK Cancel

Job Properties - Package_SSIS_Execute_Job

Select a page: General, Steps, Schedules, Alerts, Notifications, Targets

Script Help

Job step list:

St...	Name	Type	On Success	On Failure
1	P1_Transfer_Data	SQL Serv...	Quit the j...	Quit the job...
2	P2_Audit_INCR_Transfer_Data	SQL Serv...	Quit the j...	Quit the job...
3	P3_Tracability_Transfer_Data	SQL Serv...	Go to the...	Quit the job...
4	P4_Transfer_Aggregate_table	SQL Serv...	Go to the...	Quit the job...

Move step: Start step: 1:P1_Transfer_Data

New... Insert... Edit Delete

OK Cancel

Job Schedule Properties - Execute the job weekly

Name: Jobs in Schedule

Schedule type: ☒ Enabled

One-time occurrence

Date: Time:

Frequency

Occurs:

Recurs every: week(s) on

☐ Monday ☐ Wednesday ☐ Friday ☐ Saturday
☐ Tuesday ☐ Thursday ☒ Sunday

Daily frequency

☒ Occurs once at:
☐ Occurs every: hour(s)

Starting at:
Ending at:

Duration

Start date: ☐ End date: ☒ No end date:

Summary

Description:

OK Cancel Help

IV. Partie 2 : Data Warehouse Optimisation

1. Partitionnement

Le partitionnement est très nécessaire pour avoir un temps de réponse au niveau d'interrogation de la base de données, ainsi pour une meilleure performance. Pour cela on a choisi de partitionner la table SHOP_FACTS qui représente la table la plus volumineuse contenant 89171 lignes.

Ce partitionnement sera **by range** et voici ci-dessous la procédure qu'on a choisi pour procéder à ce partitionnement :

- Création des storages
- Ajout des fichiers aux storages
- Création de la fonction de partitionnement
- Création du schéma de partitionnement
- Création de la table partitionnée

2. Analysis Services

Le cube OLAP et ses dimensions est très utiles et nécessaires pour notre projet. Pour cela on l'avait créé sous Analysis Services.

Premièrement, pour pouvoir se connecter à notre base de données SQL Server, on a créé une nouvelle source de données.

Ensuite, on a créé un **Data Source View** qui a pour objectif d'établir une connexion avec la source de données et qui fournit des informations nécessaires sur la création du cube, les dimensions et les mesures associées et disponibles.

Une fois ces deux tâches sont réalisées, on peut créer notre premier cube. Ensuite on peut insérer les différentes dimensions qu'on désire visualiser.

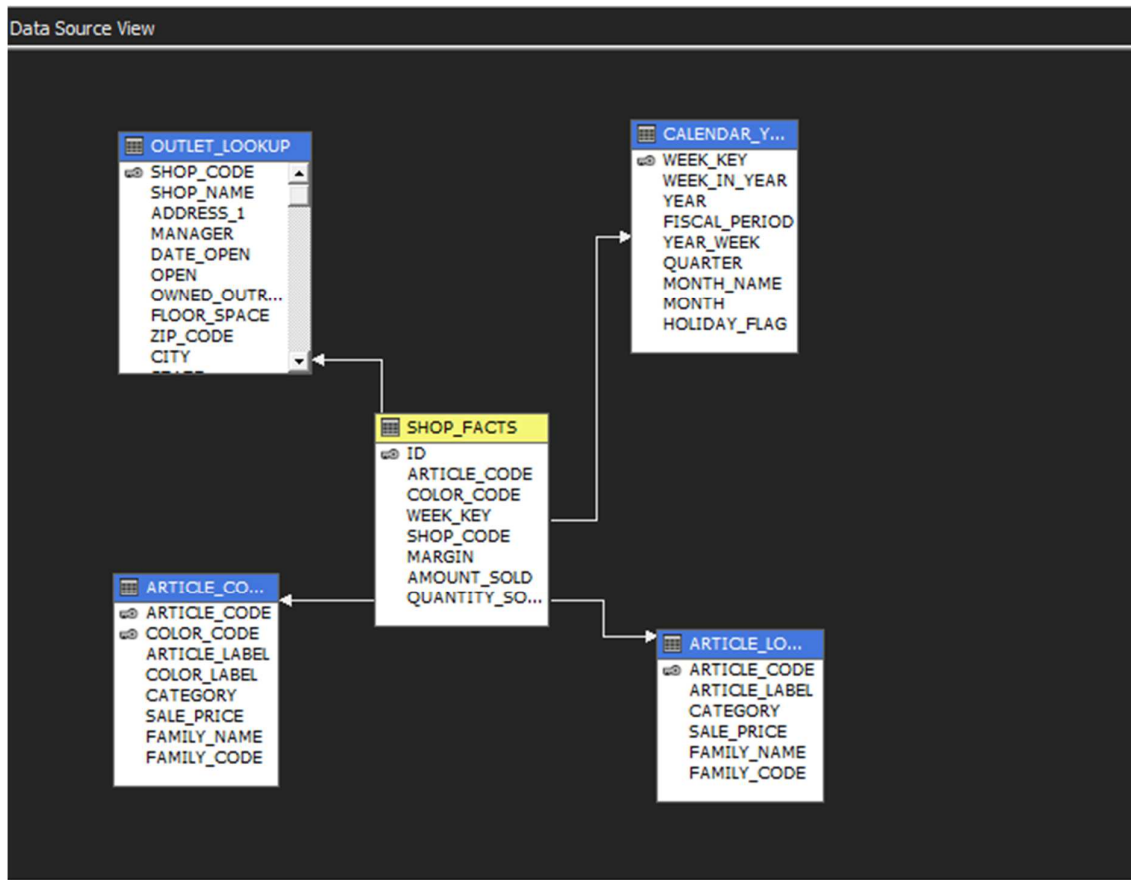
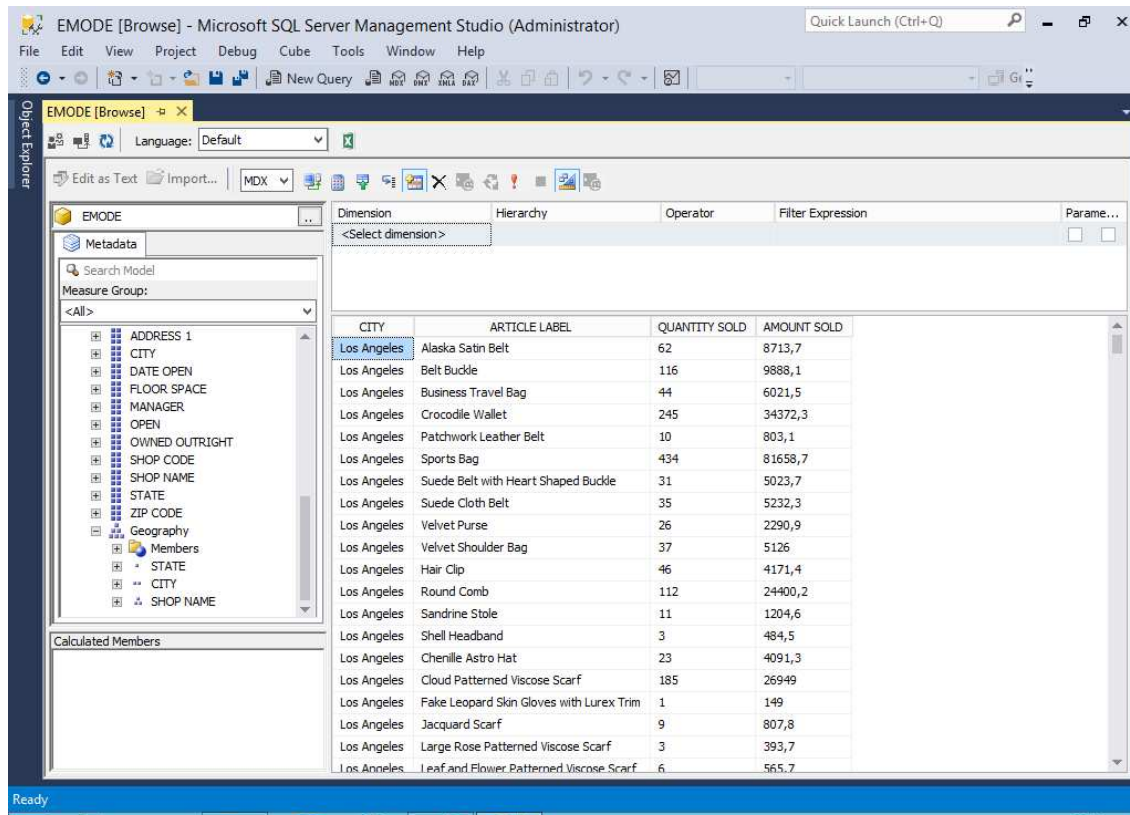


PHOTO H12RARCHIE



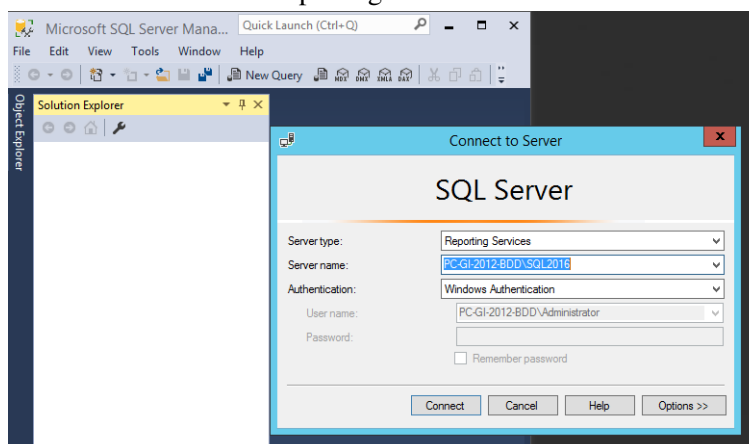
CITY	ARTICLE LABEL	QUANTITY SOLD	AMOUNT SOLD
Los Angeles	Alaska Satin Belt	62	8713,7
Los Angeles	Belt Buckle	116	9888,1
Los Angeles	Business Travel Bag	44	6021,5
Los Angeles	Crocodile Wallet	245	34372,3
Los Angeles	Patchwork Leather Belt	10	803,1
Los Angeles	Sports Bag	434	81658,7
Los Angeles	Suede Belt with Heart Shaped Budde	31	5023,7
Los Angeles	Suede Cloth Belt	35	5232,3
Los Angeles	Velvet Purse	26	2290,9
Los Angeles	Velvet Shoulder Bag	37	5126
Los Angeles	Hair Clip	46	4171,4
Los Angeles	Round Comb	112	24400,2
Los Angeles	Sandrine Stole	11	1204,6
Los Angeles	Shell Headband	3	484,5
Los Angeles	Chenille Astro Hat	23	4091,3
Los Angeles	Cloud Patterned Viscose Scarf	185	26949
Los Angeles	Fake Leopard Skin Gloves with Lurex Trim	1	149
Los Angeles	Jacquard Scarf	9	807,8
Los Angeles	Large Rose Patterned Viscose Scarf	3	393,7
Los Angeles	Leaf and Flower Patterned Viscose Scarf	6	565,7

Partie 3 : Mise en place du Reporting

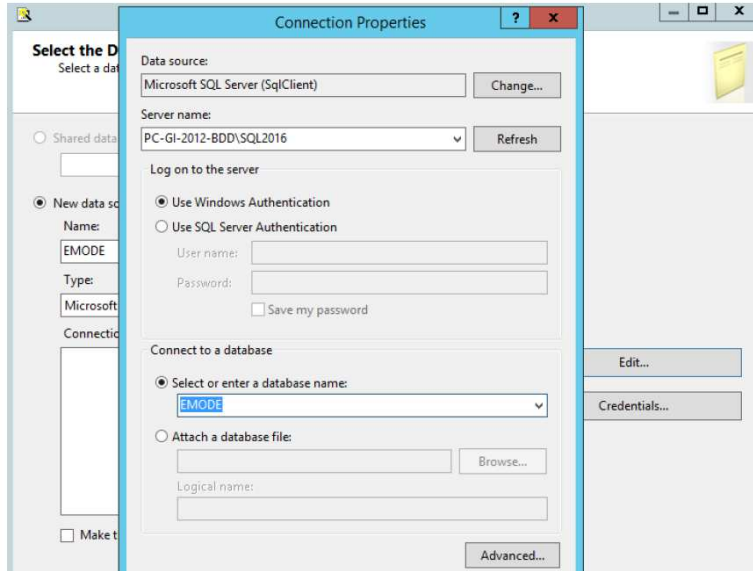
1. MS Reporting Services

Pour créer les rapports dans cette partie, on a fait les étapes suivantes

- Vérification de démarrage des services SQL Server Integration Services
- Connexion au Reporting Services



- Création d'un projet Report Server sur SSDT
- Définition de la source de données



Définition d'une dataset à l'aide de la requête SQL suivante :

SELECT

 CYL.YEAR, OL.STATE, SUM(SF.QUANTITY_SOLD) AS QUANTITY

FROM

 ARTICLE_LOOKUP AS AL

 INNER JOIN ARTICLE_COLOR_LOOKUP AS ACL

 ON AL.ARTICLE_CODE = ACL.ARTICLE_CODE

 INNER JOIN SHOP_FACTS AS SF

 ON AL.ARTICLE_CODE = SF.ARTICLE_CODE

 AND ACL.ARTICLE_CODE = SF.ARTICLE_CODE

 AND ACL.COLOR_CODE = SF.COLOR_CODE

 INNER JOIN AGG_YR_FP_YW_MO_TI_RV_MA_QT AS AGG

 INNER JOIN CALENDAR_YEAR_LOOKUP AS CYL

 ON AGG.WEEK_KEY = CYL.WEEK_KEY

 ON SF.WEEK_KEY = CYL.WEEK_KEY

 INNER JOIN OUTLET_LOOKUP AS OL

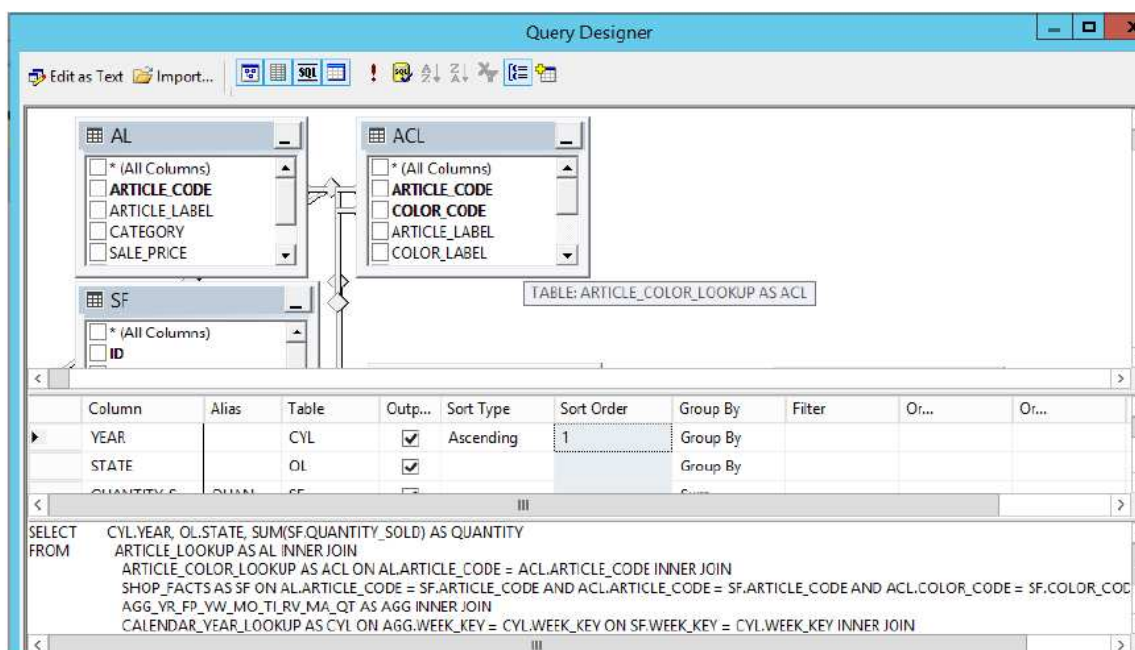
 ON SF.SHOP_CODE = OL.SHOP_CODE

 CROSS JOIN AGG_YR_FP_YW_TI_RV_MA_QT

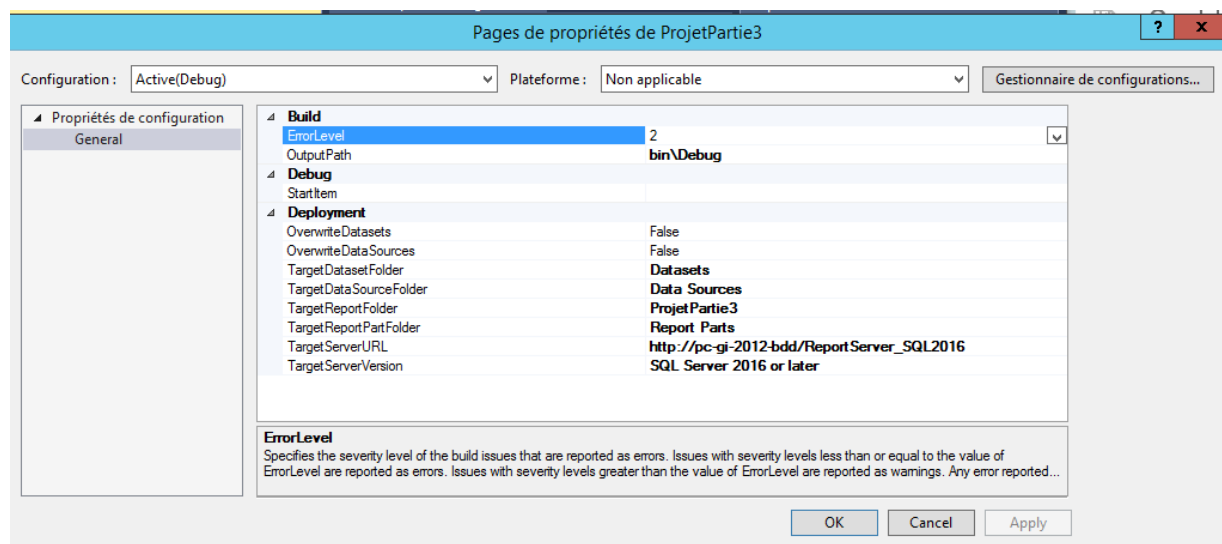
 GROUP BY CYL.YEAR, OL.STATE

 ORDER BY CYL.YEAR

Le résultat de la requête est le suivant :

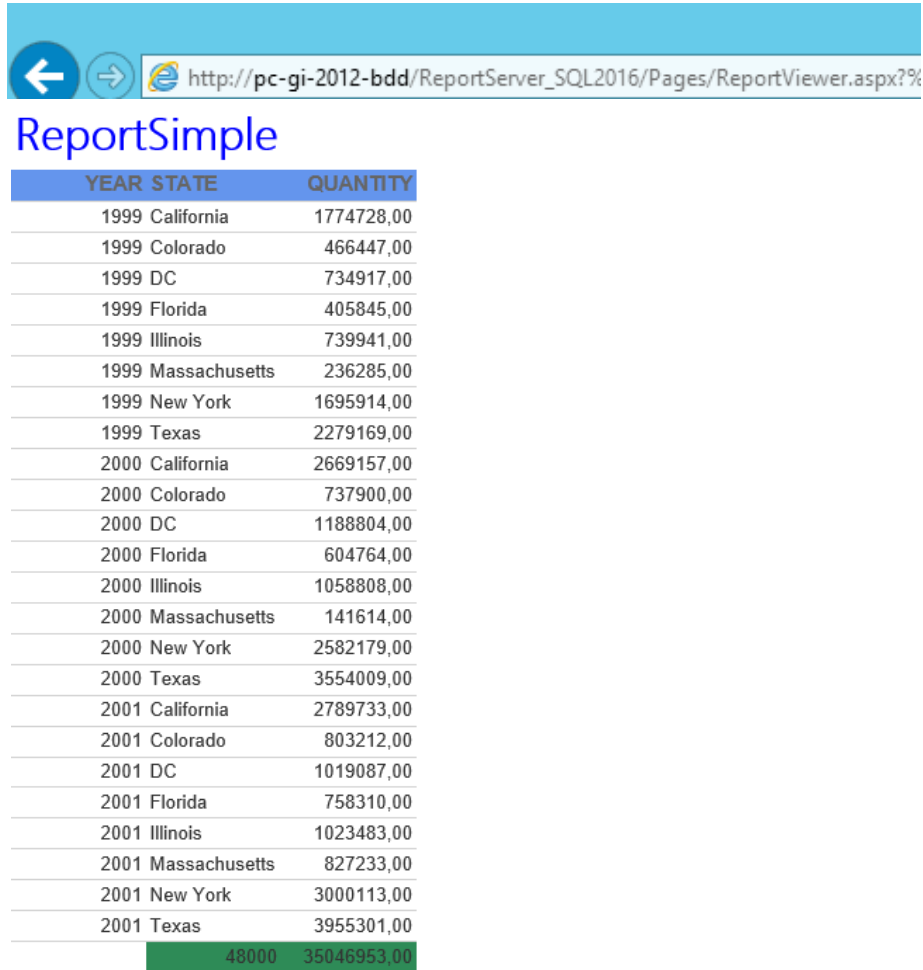


Comme ça, on peut donc créer nos rapports.



la on configure l'url pour déployer un projet.

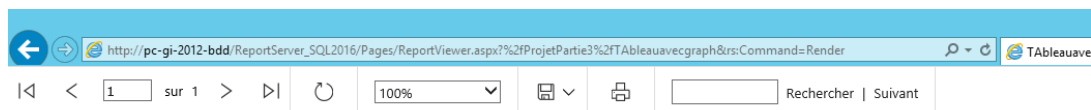
i. Tableau simple



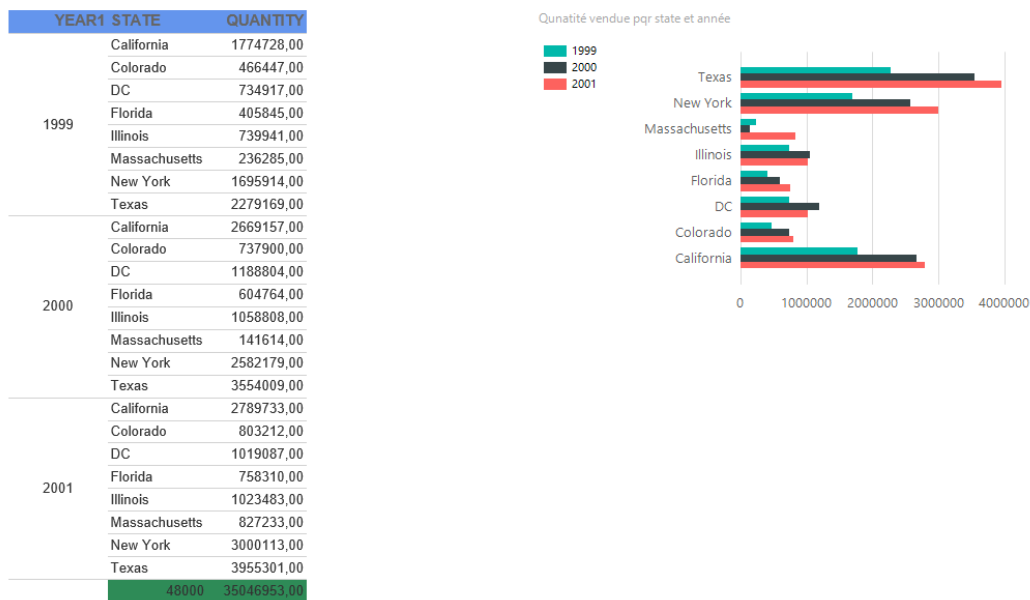
YEAR	STATE	QUANTITY
1999	California	1774728,00
1999	Colorado	466447,00
1999	DC	734917,00
1999	Florida	405845,00
1999	Illinois	739941,00
1999	Massachusetts	236285,00
1999	New York	1695914,00
1999	Texas	2279169,00
2000	California	2669157,00
2000	Colorado	737900,00
2000	DC	1188804,00
2000	Florida	604764,00
2000	Illinois	1058808,00
2000	Massachusetts	141614,00
2000	New York	2582179,00
2000	Texas	3554009,00
2001	California	2789733,00
2001	Colorado	803212,00
2001	DC	1019087,00
2001	Florida	758310,00
2001	Illinois	1023483,00
2001	Massachusetts	827233,00
2001	New York	3000113,00
2001	Texas	3955301,00
48000		35046953,00

Affichage des quantités vendues triées par année selon les états :

ii. Tableau croisé et graphique



ReportSimple



Affichage des quantités vendues par année et par état avec total et graphique associée :

2. Univers BO

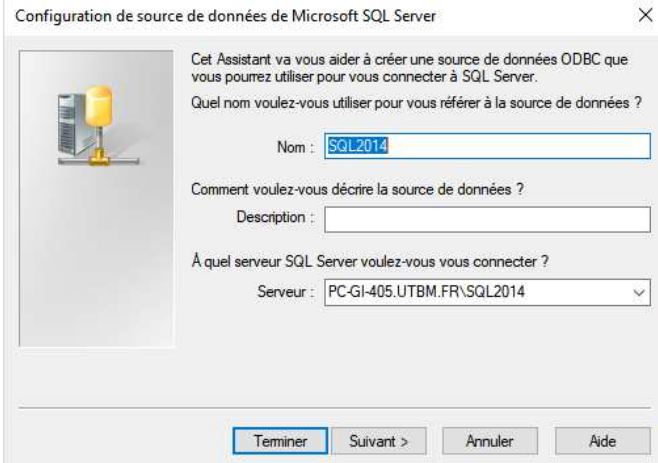
Cette partie consiste à créer un univers BO avec le modèle en étoile autour de la table SHOP_FACTS, en utilisant Universe Designer.

Pour réaliser ce modèle, nous avons importé la base de données EMODE du serveur PD-GI-405\SQL2014 en utilisant la connexion ZA_UTBM_SQL2014_EMODE.

Les étapes de configuration est le suivant :

	Nom de la source	Date de création	Propriétaire	Taille
<input checked="" type="checkbox"/>	Sources de données ODBC (32 bits)	12/04/2018 01:34	Raccourci	2 Ko
<input checked="" type="checkbox"/>	Sources de données ODBC (64 bits)	12/04/2018 01:34	Raccourci	2 Ko

Premier étape est d'ajouter un nouveau datasource 32bit.



Configuration de source de données de Microsoft SQL Server

Cet Assistant va vous aider à créer une source de données ODBC que vous pourrez utiliser pour vous connecter à SQL Server.

Quel nom voulez-vous utiliser pour vous référer à la source de données ?

Nom :

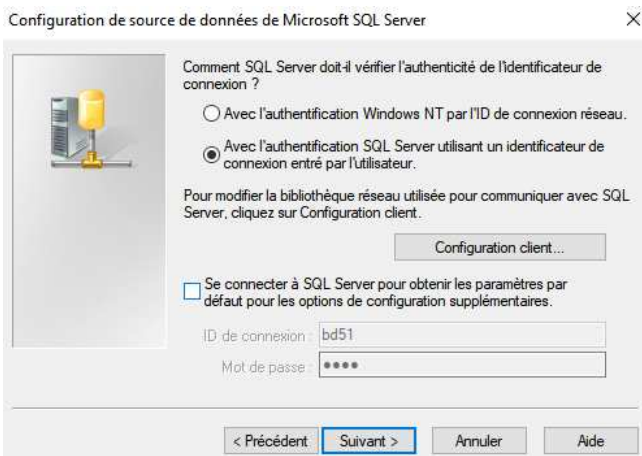
Comment voulez-vous décrire la source de données ?

Description :

À quel serveur SQL Server voulez-vous vous connecter ?

Serveur :

On a configure le nom et le serveur comme dans la figure



Configuration de source de données de Microsoft SQL Server

Comment SQL Server doit-il vérifier l'authenticité de l'identificateur de connexion ?

☐ Avec l'authentification Windows NT par l'ID de connexion réseau.

☒ Avec l'authentification SQL Server utilisant un identificateur de connexion entré par l'utilisateur.

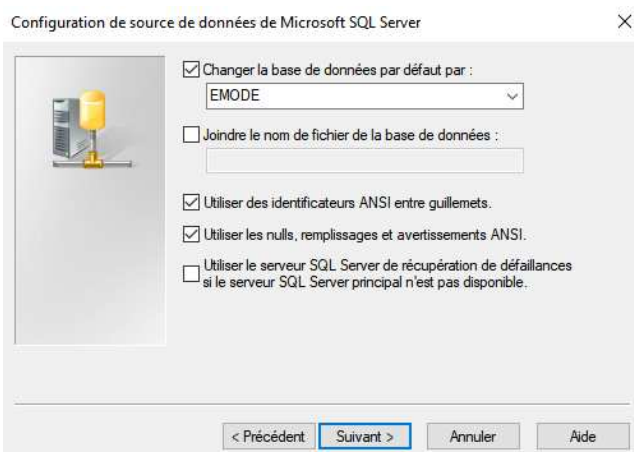
Pour modifier la bibliothèque réseau utilisée pour communiquer avec SQL Server, cliquez sur Configuration client.

☐ Se connecter à SQL Server pour obtenir les paramètres par défaut pour les options de configuration supplémentaires.

ID de connexion :

Mot de passe :

Puis la configuration de l'utilisateur bd51 avec mots de passe bd51.



Configuration de source de données de Microsoft SQL Server

☒ Changer la base de données par défaut par :

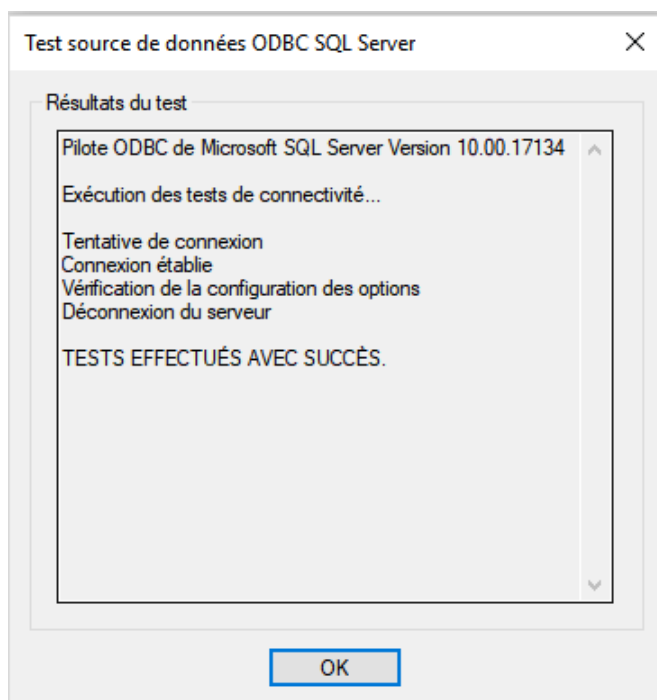
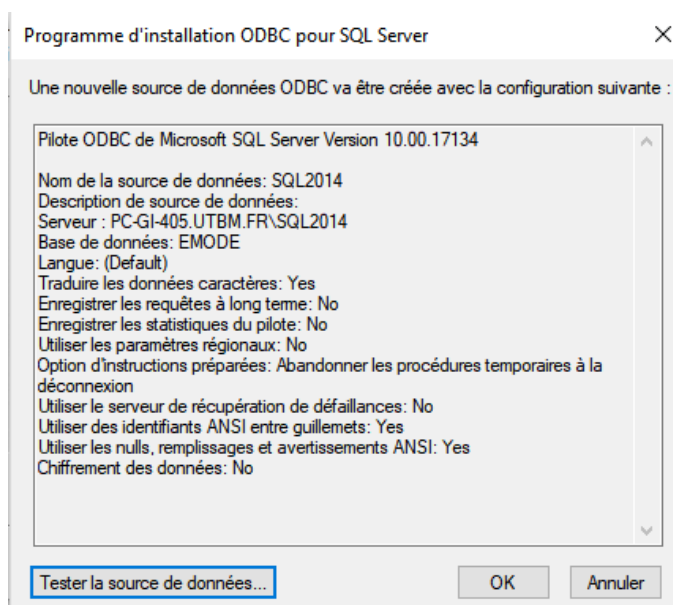
☐ Joindre le nom de fichier de la base de données :

☒ Utiliser des identificateurs ANSI entre guillemets.

☒ Utiliser les nulls, remplissages et avertissements ANSI.

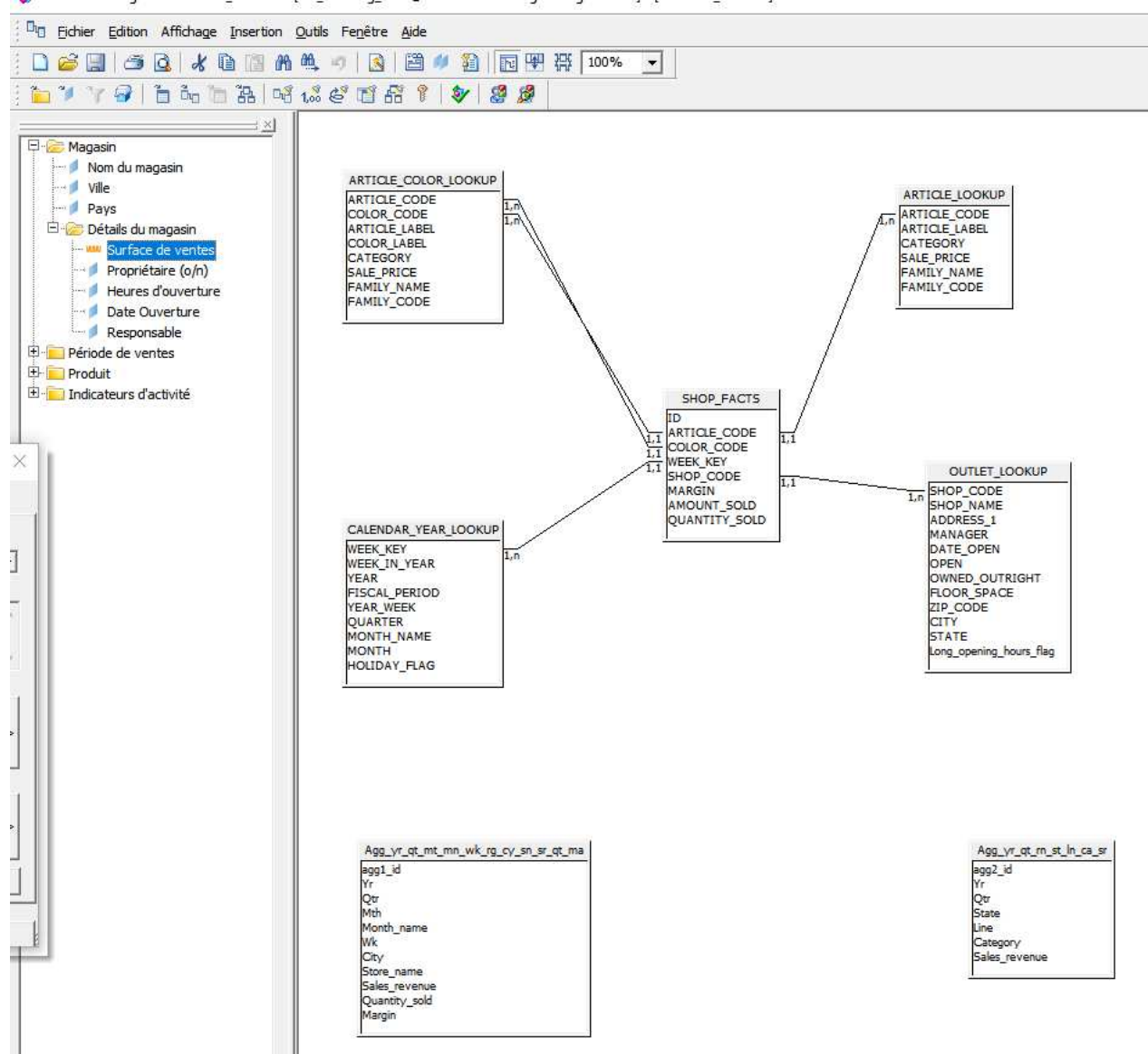
☐ Utiliser le serveur SQL Server de récupération de défaillances si le serveur SQL Server principal n'est pas disponible.

Après on choisit la base de donnée par défaut.



Le test de connexion effectue avec succès.

Après avoir configuré le projet comme demandé nous avons implémenté le modèle en étoile pour obtenir le résultat final suivant:



2. Web Intelligence

Premier document : B01SalesRevenueAnalysis.wid

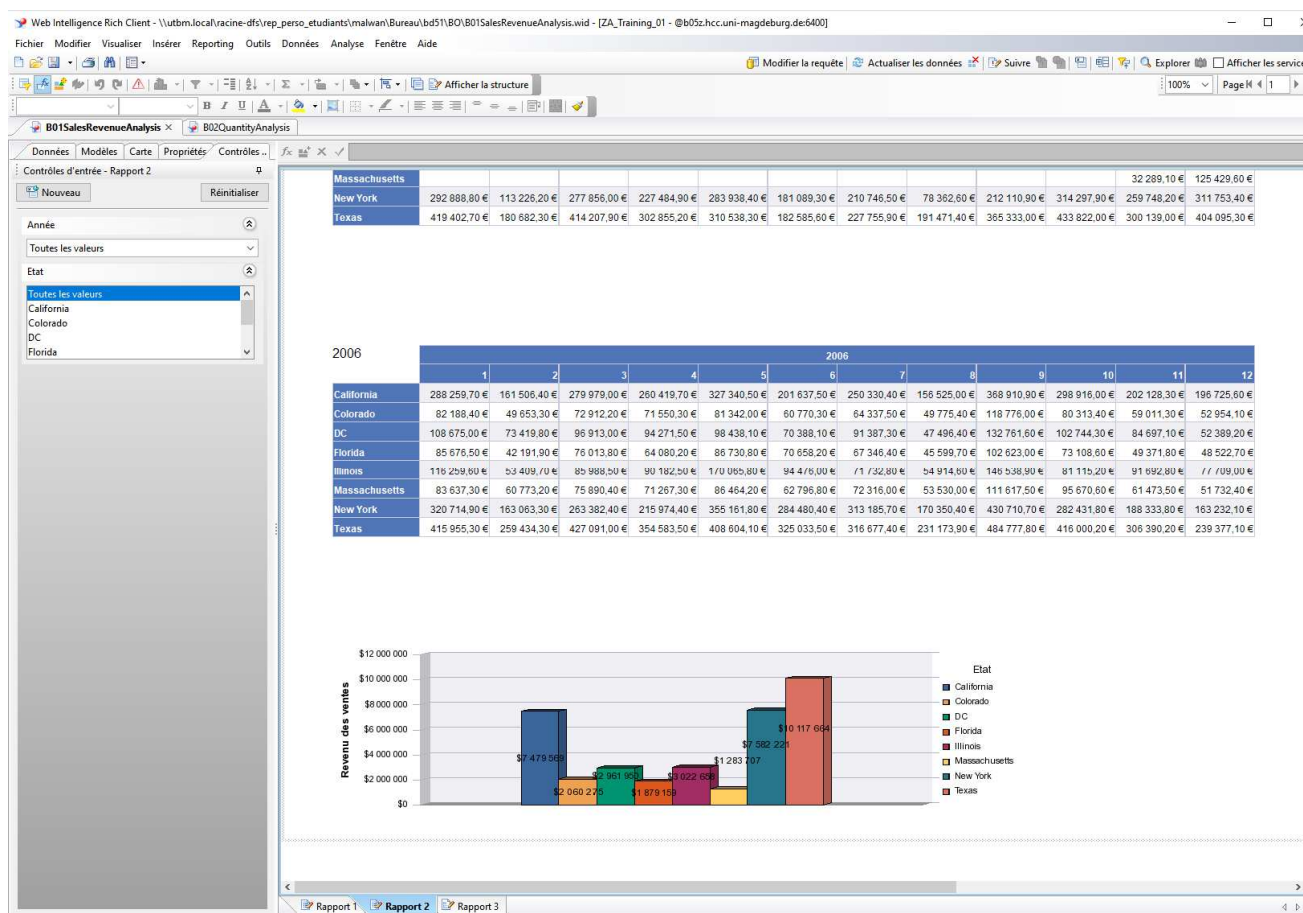
i. Premier rapport

Rapport selon l'année et l'état (Tableau croisé dynamique)

2004	2004											
	1	2	3	4	5	6	7	8	9	10	11	12
California	178 186,40 €	120 204,50 €	220 829,50 €	185 423,30 €	163 346,30 €	92 724,60 €	115 918,10 €	36 010,70 €	242 379,80 €	141 647,80 €	100 113,60 €	107 426,20 €
Colorado	41 686,80 €	35 181,30 €	54 928,80 €	55 378,40 €	43 661,00 €	30 036,90 €	33 139,60 €	9 012,10 €	43 469,20 €	38 580,90 €	27 832,90 €	35 393,60 €
DC	74 120,30 €	48 128,20 €	86 075,90 €	82 401,20 €	60 683,00 €	36 778,90 €	48 804,30 €	14 281,20 €	68 601,20 €	78 923,20 €	40 881,40 €	53 531,70 €
Florida	55 731,50 €	31 099,40 €	50 698,80 €	41 044,10 €	49 189,70 €	30 936,50 €	23 097,90 €	9 698,20 €	18 130,10 €	28 762,60 €	24 240,20 €	43 356,10 €
Illinois	101 985,00 €	63 674,10 €	90 794,70 €	87 755,90 €	95 828,90 €	57 563,90 €	42 213,90 €	11 402,80 €	53 388,80 €	54 456,60 €	37 917,40 €	40 932,20 €
Massachusetts	38 797,60 €	27 822,70 €	25 975,20 €	24 839,00 €	28 984,50 €	17 079,20 €	10 994,50 €	1 071,00 €			4 581,70 €	58 673,30 €
New York	222 473,80 €	124 299,80 €	209 209,50 €	174 392,10 €	179 540,80 €	126 028,60 €	128 788,90 €	39 452,60 €	88 872,40 €	138 460,70 €	118 595,40 €	117 581,20 €
Texas	290 559,80 €	179 663,20 €	288 572,70 €	244 025,80 €	244 380,90 €	126 669,90 €	122 946,30 €	52 827,80 €	153 339,30 €	174 374,60 €	129 861,60 €	192 455,50 €

ii. Second rapport

Rapport selon l'année l'état avec graphique et somme et avec possibilité de filtre :



iii. Troisième rapport :

Page de couverture du document :

Analyse mutple des ventes

En

1. Analyse mensuelle des ventes par tableau croisé

2. Analyse mensuelle des ventes par graphique

Second document : B02QuantityAnalysis.wid

i. Premier rapport

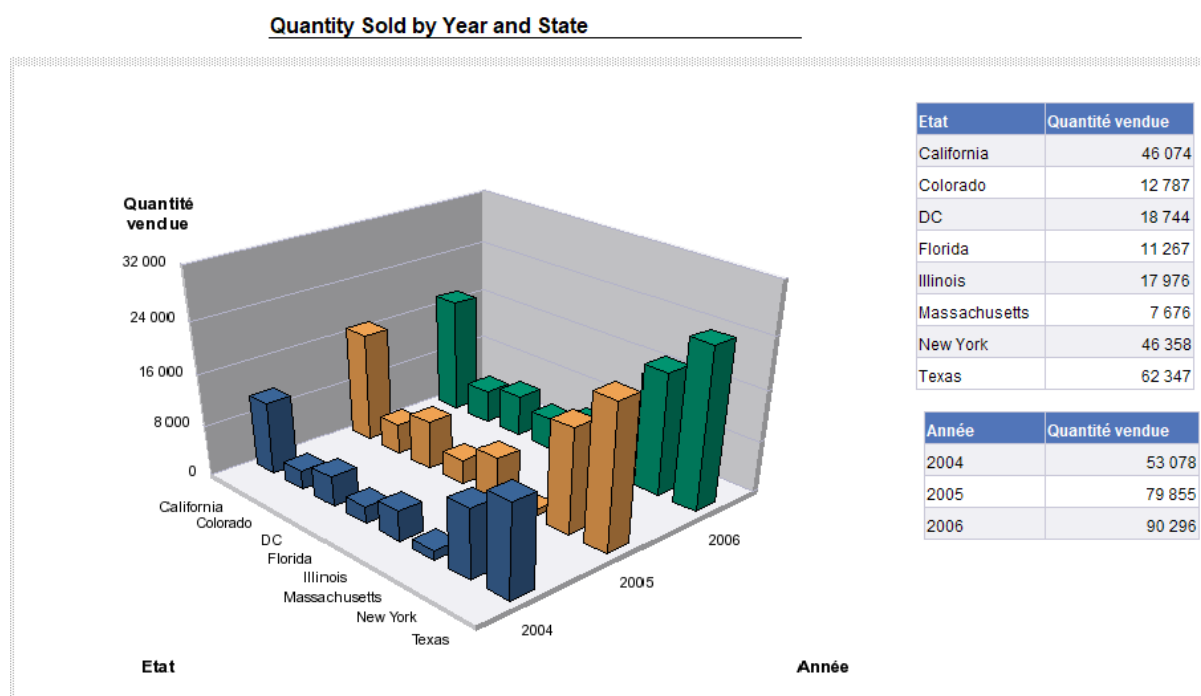
Quantité vendue selon l'année et le lieu avec total :

Quantity Sold by Year and State

Année	Etat	Quantité vendue
2004	California	11 304
	Colorado	2 971
	DC	4 681
	Florida	2 585
	Illinois	4 713
	Massachusetts	1 505
	New York	10 802
	Texas	14 517
	S/Total :	53 078

ii. Second rapport

Quantité vendue par état et par an avec histogramme 3D



iii. Troisième rapport

Quantité vendu par état avec son pourcentage et pie chart avec filtres :



3. Reporting avec Excel

Dans cette partie, la base de données a été importée à partir de la base EMODE sur le serveur PC-GI-405 (SQL Server).

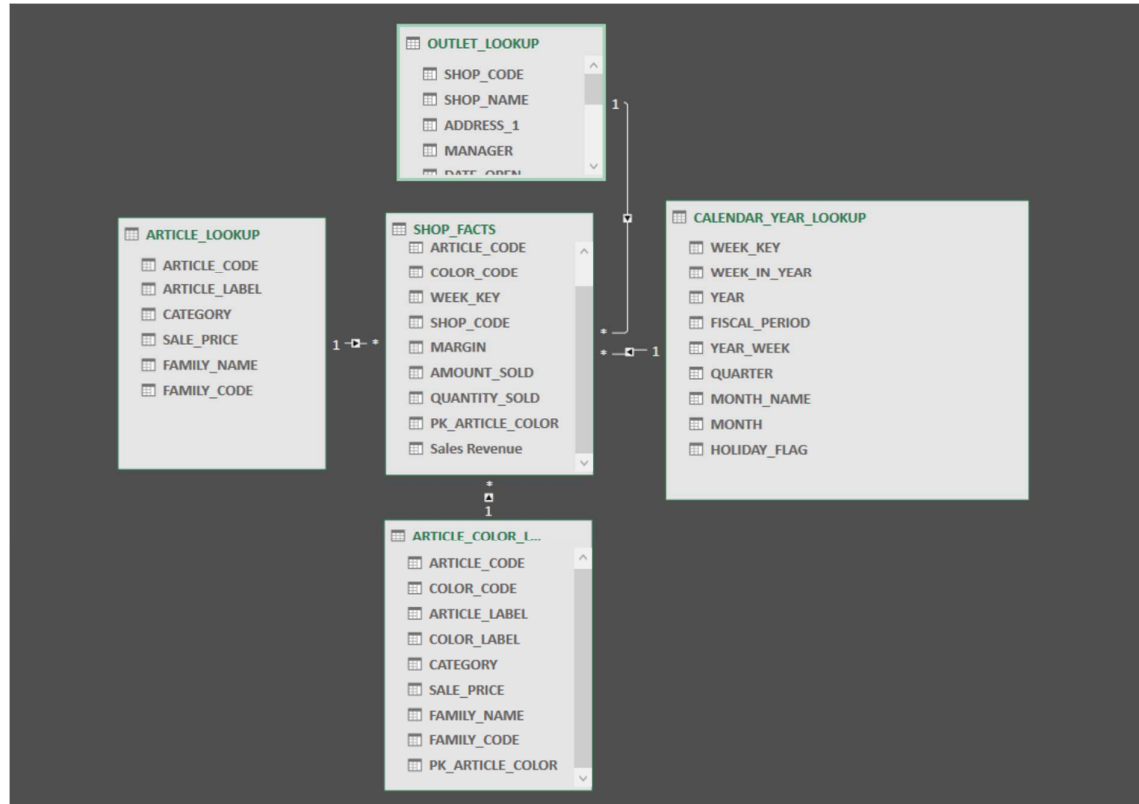
Nom du serveur: PC-GI-405\SQL2014

Nom de la base de données: EMODE

Informations de connexion: bd51/bd51

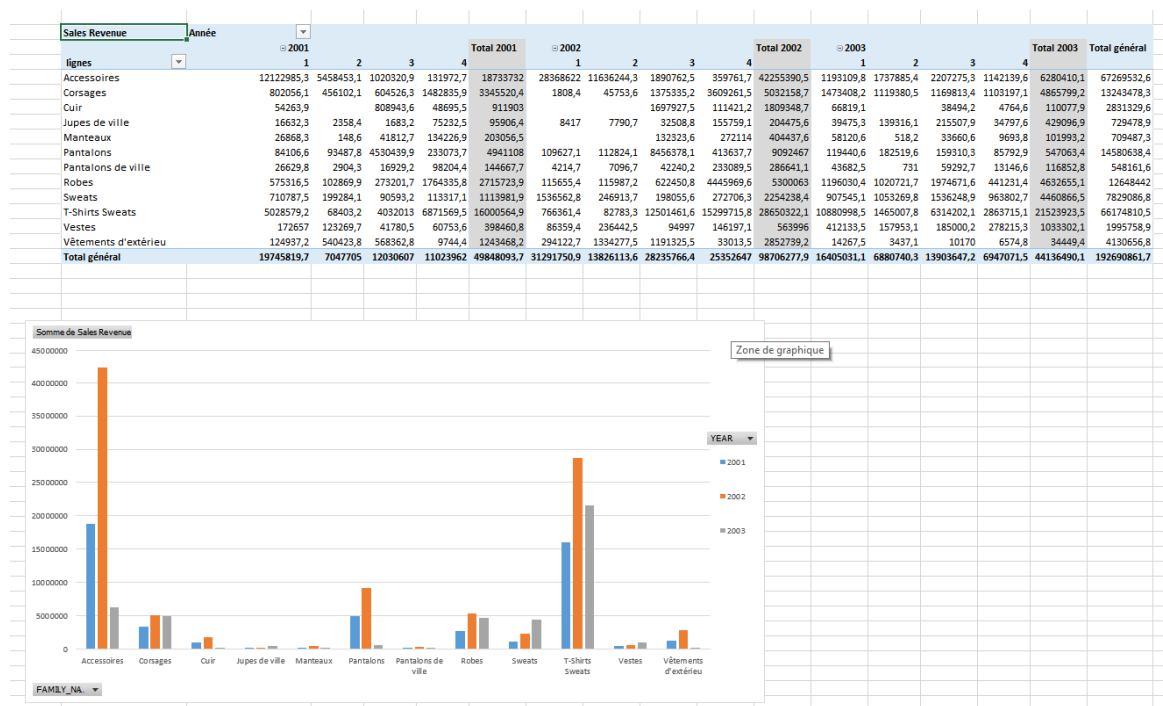
i. Modèle en étoile

La figure ci-dessous représente le modèle en étoile défini dans Power Pivot de Excel :

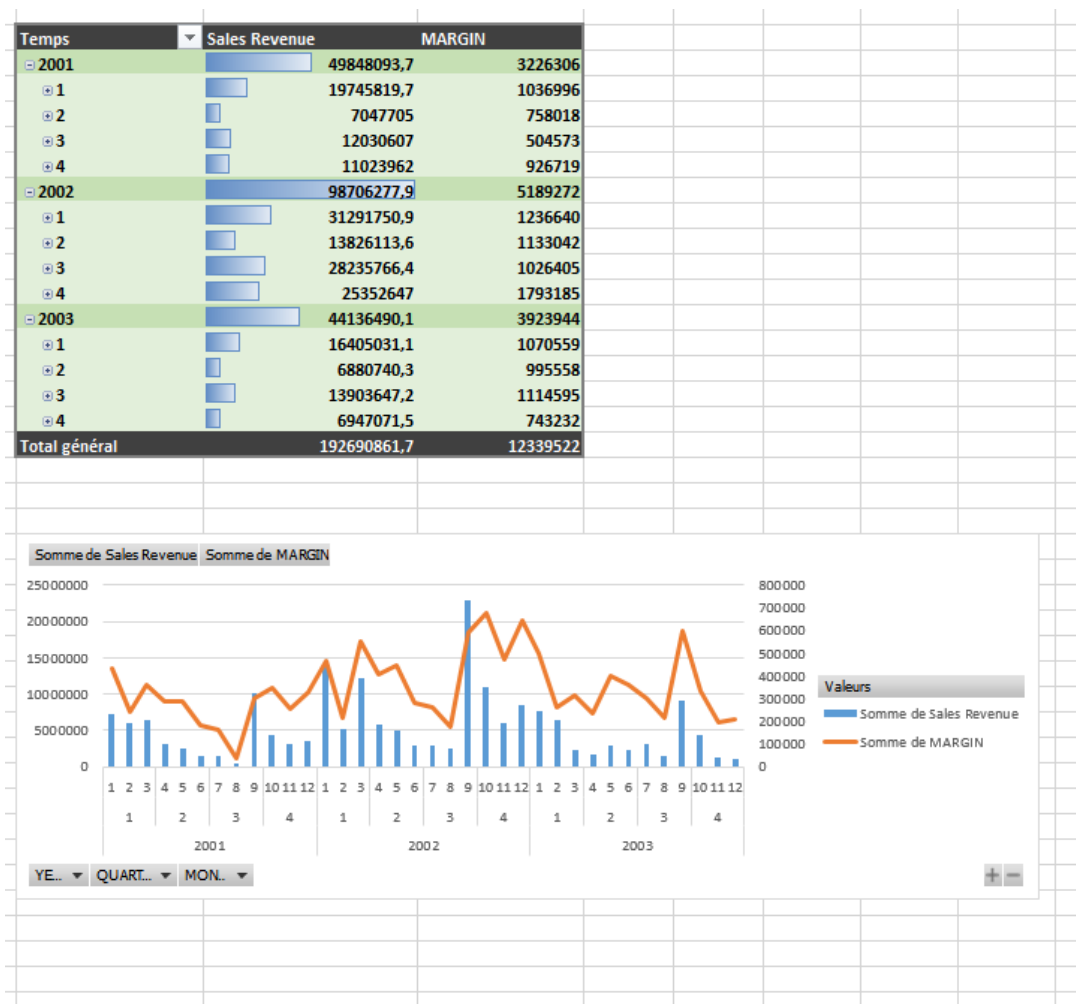


ii. Premier rapport : tableau croisé avec graphique

L'affichage des données est dynamique selon si on déroule ou non les détails à côté des différentes années.



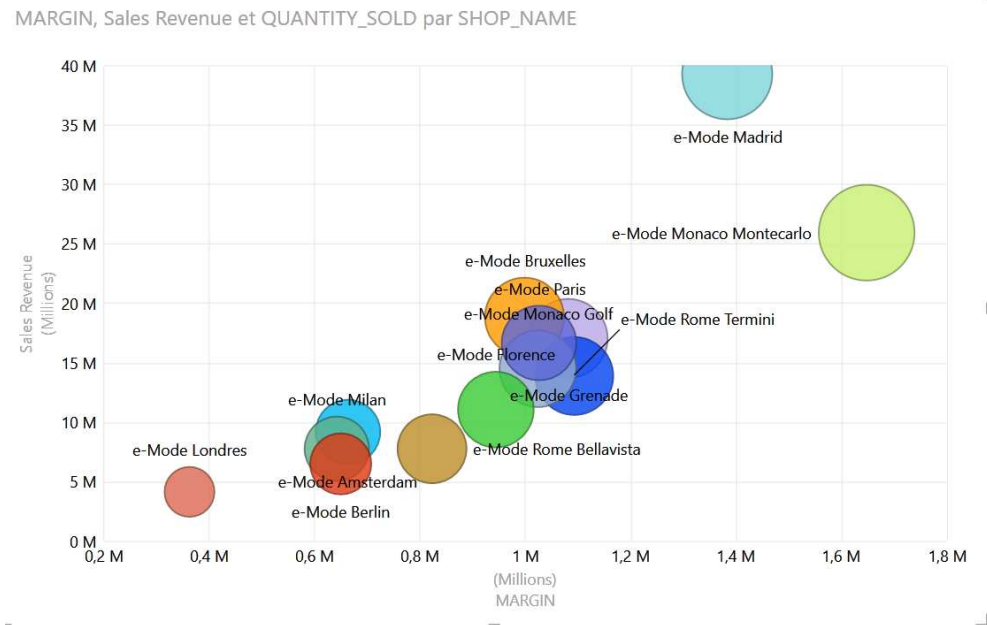
iii. Second rapport : tableau croisé avec graphique



iv. Troisième rapport : Power View

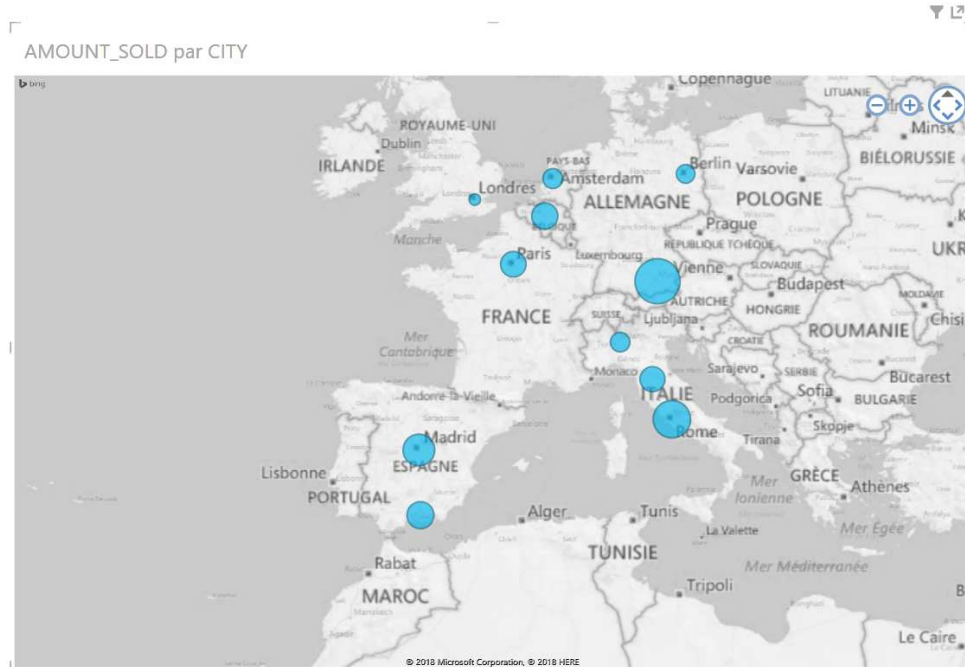
Animation de l'évolution du CA et marge sur trois années par Magasin :

Evolution du CA/Marge sur 3 années par magasin

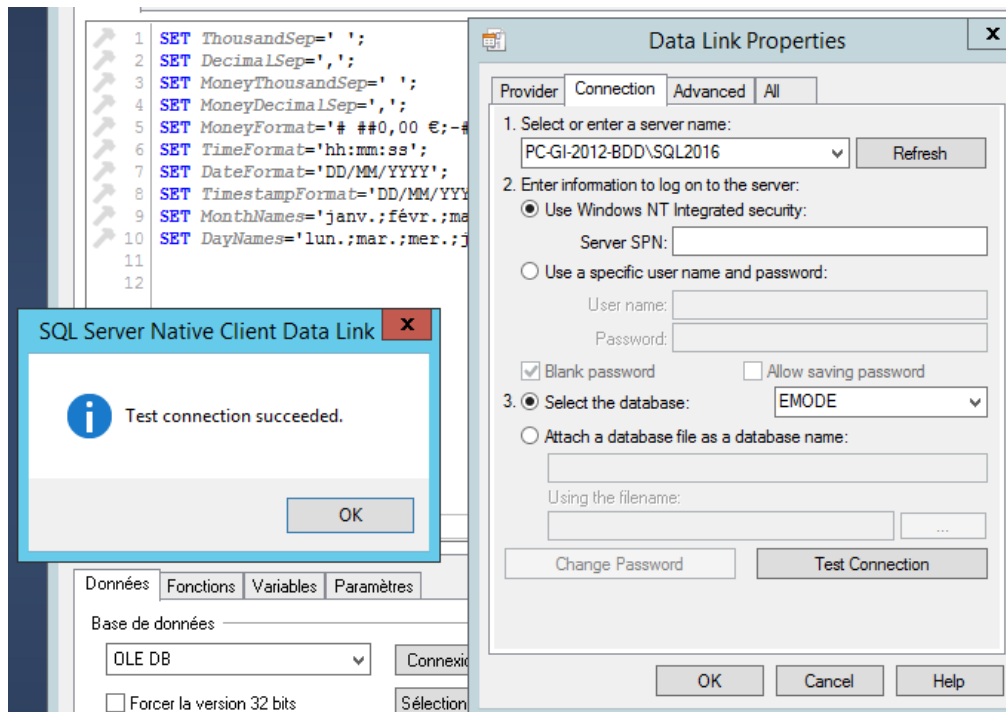


v. Quatrième rapport

Affichage des ventes par ville sur une carte :

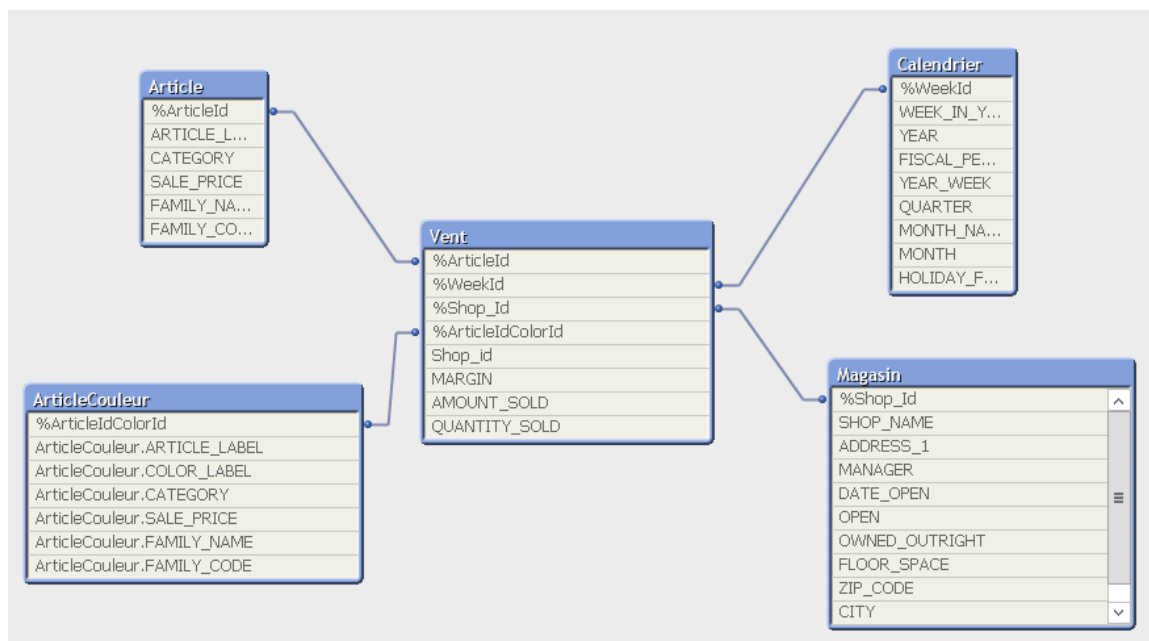


4. Reporting avec QlikView

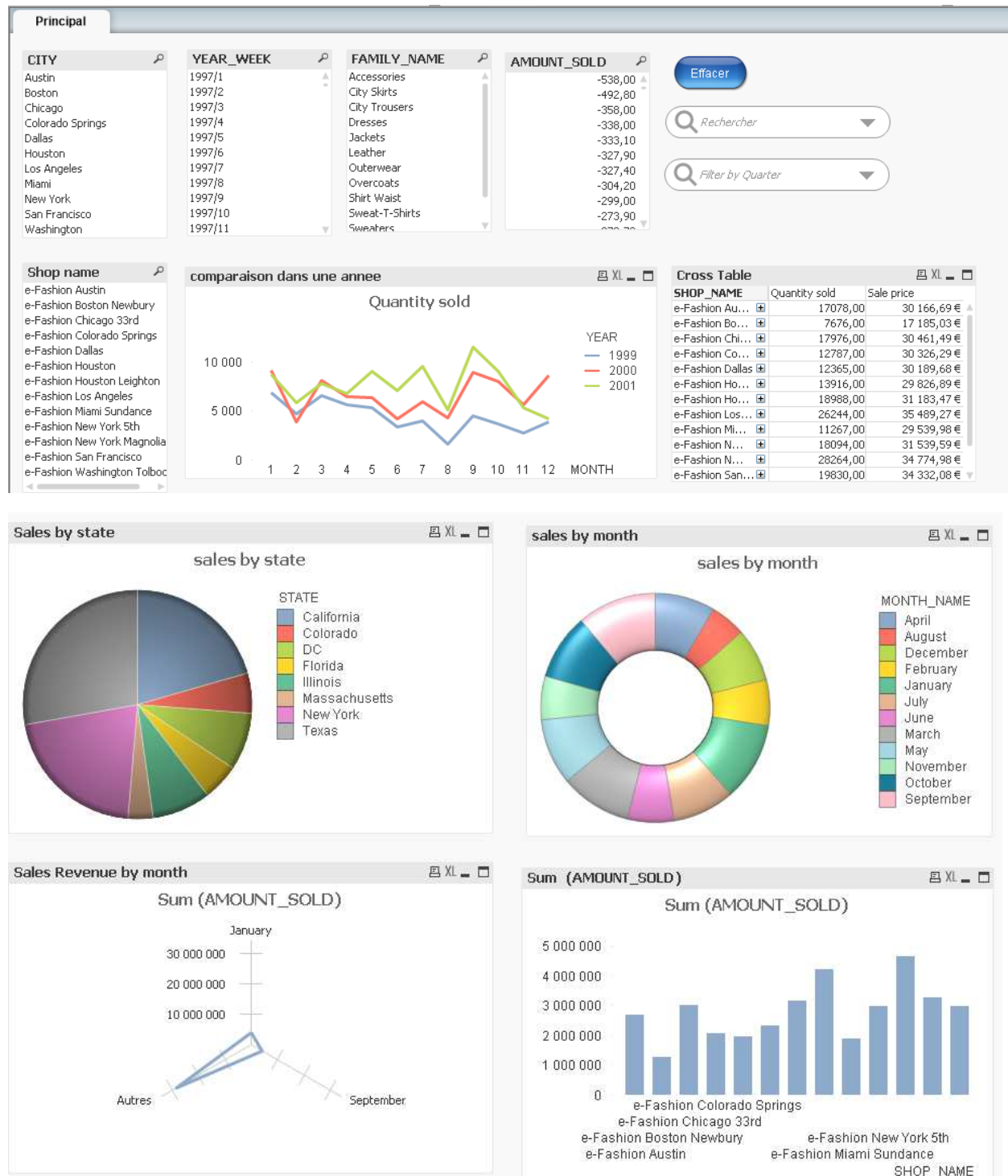


Ajout un datasource avec la configuration ci-dessus.

i. Le model étoile de la base de donnée EMODE



Tableaux de bord avec plusieurs tableaux et diagrammes :



V. Conclusion

Ce projet nous a permis d'acquérir de nouvelles compétences et d'enrichir nos connaissances acquises durant le cours et les Tps, soit au niveau de la rédaction des requêtes, les fonctions ETL, le transfert des données, l'optimisation de l'entrepôt de données, le partitionnement des tables, la création du cube et des dimensions pour les tables de faits et de dimensions, et finalement au niveau du reporting et la visualisation des données en créant des Dashboard dynamique.

En plus de l'expérience qu'on a pu acquérir durant ce projet, ce travail était très enrichissant pour nous au niveau du travail en équipe, car il nous a permis bien sure de partager les idées et de choisir la solution la plus adapté à notre problématique malgré les difficultés rencontrées.

Annexe

Table de Figures

1. Qualité des données

--Liste des tables, description des tables

```
SELECT * FROM TAB ;
```

```
DESC ARTICLE_COLOR_LOOKUP;
```

```
DESC ARTICLE_LOOKUP;
```

```
DESC ARTICLE_LOOKUP_CRITERIA;
```

```
DESC CALENDAR_YEAR_LOOKUP;
```

```
DESC OUTLET_LOOKUP;
```

```
DESC PRODUCT_PROMOTION_FACTS;
```

```
DESC PROMOTION_LOOKUP;
```

```
DESC SHOP_FACTS;
```

- Nombre d'élément de chaque table

```
SELECT COUNT(CONCAT(ARTICLE_CODE,COLOR_CODE)) AS "NB ARTICLECODE/COLOR CODE"
```

```
FROM ARTICLE_COLOR_LOOKUP;
```

```
SELECT COUNT(*) AS "NB ARTICLE CODE" FROM ARTICLE_LOOKUP;
```

```
SELECT COUNT(*) AS "NB ID ARTICLE CRITERIA" FROM ARTICLE_LOOKUP_CRITERIA;
```

```
SELECT COUNT(*) AS "NB WEEK_KEY" FROM CALENDAR_YEAR_LOOKUP;
```

```
SELECT COUNT (*) AS "NB SHOP CODE" FROM OUTLET_LOOKUP;
```

```
SELECT COUNT(*) AS "NB PROMOTION PRODUCT" FROM PRODUCT_PROMOTION_FACTS;
```

```
SELECT COUNT(*) AS "NB PROMOTION_LOOKUP" FROM PROMOTION_LOOKUP;
```

```
SELECT COUNT(*) AS "NB SHOP_FACTS" FROM SHOP_FACTS;
```

- Clef primaires en double de la table ARTICLE_COLOR_LOOKUP

```
SELECT
```

```
    ACL.ARTICLE_CODE AS "ARTICLE CODE"
```

```
    , ACL.COLOR_CODE AS "COLOR CODE"
```

```
    , COUNT(CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)) AS "NB PK ARTICLE CODE + COLOR CODE"
```

```
FROM
```

```
    ARTICLE_COLOR_LOOKUP ACL
```

GROUP BY

ACL.ART

ICLE_CODE

, ACL.COLOR_CODE

, CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)

HAVING COUNT(CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)) > 1

ORDER BY

ACL.ARTICLE_CODE,ACL.COLOR_CODE ASC;

-- Clefs étrangères manquantes de la table ARTICLE_COLOR_LOOKUP

SELECT

ACL.ARTICLE_CODE AS "ARTICLE CODE"

FROM

ARTICLE_COLOR_LOOKUP ACL

WHERE

ACL.ARTICLE_CODE NOT IN (SELECT

ALO.ARTICLE_CODE

FROM

ARTICLE_LOOKUP ALO)

ORDER BY

ACL.ARTICLE_CODE ASC;

- Clef primaires en double de la table ARTICLE_LOOKUP

SELECT

A.ARTICLE_CODE

,COUNT(A.ARTICLE_CODE) AS "NB PK"

FROM

ARTICLE_LOOKUP A

HAVING COUNT(A.ARTICLE_CODE) > 1

GROUP BY

A.ARTICLE_CODE

ORDER BY

A.ARTICLE_CODE ASC ;

-- Clef primaires en double de la table ARTICLE_LOOKUP_CRITERIA

SELECT

A.ID AS "ID"
,COUNT(A.ID) AS "NB ID"

FROM

ARTICLE_LOOKUP_CRITERIA A
HAVING COUNT(A.ID) > 1

GROUP BY

A.ID;

-- Clef primaires en double de la table CALENDAR_YEAR_LOOKUP

SELECT

C.WEEK_KEY

FROM

CALENDAR_YEAR_LOOKUP C

GROUP BY

C.WEEK_KEY
HAVING COUNT(C.WEEK_KEY) > 1;

- Clef primaires en double de la table OUTLET_LOOKUP

SELECT

O.SHOP_CODE AS "SHOP CODE"
,COUNT(O.SHOP_CODE) AS "NB SHOP CODE"

FROM

OUTLET_LOOKUP O

GROUP BY

O.SHOP_CODE
HAVING COUNT(O.SHOP_CODE) > 1;

- Clef primaires en double de la table SHOP_FACTS

SELECT

S.ID AS "ID"
,COUNT(S.ID) AS "NB ID"

```
FROM
    SHOP_FACTS S
GROUP BY
    S.ID
HAVING COUNT(S.ID) > 1;
-- Clefs étrangères manquantes de la table SHOP_FACTS
--WEEK_KEY
SELECT
    S.ID "ID SHOP FACTS"
    ,S.WEEK_KEY AS "WEEK KEY"
FROM
    SHOP_FACTS S
WHERE
    S.WEEK_KEY NOT IN (SELECT
        DISTINCT C.WEEK_KEY
        FROM
            CALENDAR_YEAR_LOOKUP C)
ORDER BY
    S.ID ASC;
--ARTICLE_CODE
SELECT
    S.ID "ID SHOP FACTS"
    ,S.ARTICLE_CODE AS "ARTICLE CODE"
FROM
    SHOP_FACTS S
WHERE
    S.ARTICLE_CODE NOT IN (
        SELECT
            DISTINCT A.ARTICLE_CODE
        FROM
            ARTICLE_LOOKUP A)
```

```
ORDER BY
    S.ID ASC;
-- COLOR_CODE
SELECT
    S.ID "ID SHOP FACTS"
    ,S.COLOR_CODE AS "COLOR CODE"
FROM
    SHOP_FACTS S
WHERE
    S.COLOR_CODE NOT IN (
        SELECT
            DISTINCT A.COLOR_CODE
        FROM
            ARTICLE_COLOR_LOOKUP A)
    ORDER BY
        S.ID ASC;
```

a) Create reject table

```
CREATE TABLE [dbo].[ARTICLE_COLOR_LOOKUP_TRASH](
    [ARTICLE_CODE] [numeric](6, 0) NOT NULL,
    [COLOR_CODE] [numeric](4, 0) NOT NULL,
    [ARTICLE_LABEL] [varchar](45) NULL,
    [COLOR_LABEL] [varchar](30) NULL,
    [CATEGORY] [varchar](25) NULL,
    [SALE_PRICE] [numeric](8, 2) NULL,
    [FAMILY_NAME] [varchar](30) NULL,
    [FAMILY_CODE] [varchar](3) NULL
)
GO
```

```
CREATE TABLE [dbo].[ARTICLE_LOOKUP_TRASH](
    [ARTICLE_CODE] [numeric](6, 0) NOT NULL,
```

```
[ARTICLE_LABEL] [varchar](45) NULL,  
[CATEGORY] [varchar](25) NULL,  
[SALE_PRICE] [numeric](8, 2) NULL,  
[FAMILY_NAME] [varchar](20) NULL,  
[FAMILY_CODE] [varchar](3) NULL  
)  
GO
```

```
CREATE TABLE [dbo].[CALENDAR_YEAR_LOOKUP_TRASH](  
    [WEEK_KEY] [numeric](3, 0) NOT NULL,  
    [WEEK_IN_YEAR] [numeric](2, 0) NULL,  
    [YEAR] [numeric](4, 0) NULL,  
    [FISCAL_PERIOD] [varchar](4) NULL,  
    [YEAR_WEEK] [varchar](7) NULL,  
    [QUARTER] [numeric](1, 0) NULL,  
    [MONTH_NAME] [varchar](10) NULL,  
    [MONTH] [numeric](2, 0) NULL,  
    [HOLIDAY_FLAG] [varchar](1) NULL  
)  
GO
```

```
CREATE TABLE [dbo].[OUTLET_LOOKUP_TRASH](  
    [SHOP_NAME] [varchar](30) NOT NULL,  
    [ADDRESS_1] [varchar](20) NULL,  
    [MANAGER] [varchar](10) NULL,  
    [DATE_OPEN] [datetime] NULL,  
    [OPEN] [varchar](1) NULL,  
    [OWNED_OUTRIGHT] [varchar](1) NULL,  
    [FLOOR_SPACE] [numeric](4, 0) NULL,  
    [ZIP_CODE] [varchar](6) NULL,  
    [CITY] [varchar](20) NULL,
```

```
[STATE] [varchar](20) NULL,  
[SHOP_CODE] [numeric](3, 0) NOT NULL  
)  
GO
```

```
CREATE TABLE [dbo].[SHOP_FACTS_TRASH](  
    [ID] [numeric](5, 0) NOT NULL,  
    [ARTICLE_CODE] [numeric](6, 0) NULL,  
    [COLOR_CODE] [numeric](4, 0) NULL,  
    [WEEK_KEY] [numeric](3, 0) NULL,  
    [SHOP_CODE] [numeric](3, 0) NULL,  
    [MARGIN] [numeric](18, 0) NULL,  
    [AMOUNT_SOLD] [numeric](13, 2) NULL,  
    [QUANTITY_SOLD] [numeric](13, 2) NULL  
)  
GO
```

b) Fill reject table

-- remplissage des tables de rejet

-- Table de rejet des clés primaires non uniques de la table ARTICLE_COLOR_LOOKUP

```
INSERT INTO [dbo].[ARTICLE_COLOR_LOOKUP_TRASH] (  
    ARTICLE_CODE  
    , COLOR_CODE  
    , ARTICLE_LABEL  
    , COLOR_LABEL  
    , CATEGORY  
    , SALE_PRICE  
    , FAMILY_NAME  
    , FAMILY_CODE)  
SELECT  
    ACL.ARTICLE_CODE  
    , ACL.COLOR_CODE
```



```
, ACL.ARTICLE_LABEL
, ACL.COLOR_LABEL
, ACL.CATEGORY
, ACL.SALE_PRICE
, ACL.FAMILY_NAME
, ACL.FAMILY_CODE
FROM
    ARTICLE_COLOR_LOOKUP ACL
WHERE
    CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE) IN(
        CONCAT(ACL2.ARTICLE_CODE,ACL2.COLOR_CODE)
    )
FROM
    ARTICLE_COLOR_LOOKUP ACL2
GROUP BY
    CONCAT(ACL2.ARTICLE_CODE,ACL2.COLOR_CODE)
HAVING
    COUNT(CONCAT(ACL2.ARTICLE_CODE,ACL2.COLOR_CODE))>1;
-- Table de rejet des clés étrangères manquantes de la table ARTICLE_COLOR_LOOKUP
INSERT INTO [dbo].[ARTICLE_COLOR_LOOKUP_TRASH] (
    ARTICLE_CODE
    , COLOR_CODE
    , ARTICLE_LABEL
    , COLOR_LABEL
    , CATEGORY
    , SALE_PRICE
    , FAMILY_NAME
    , FAMILY_CODE)
SELECT
    ACL.ARTICLE_CODE
    , ACL.COLOR_CODE
    , ACL.ARTICLE_LABEL
```

```
, ACL.COLOR_LABEL
, ACL.CATEGORY
, ACL.SALE_PRICE
, ACL.FAMILY_NAME
, ACL.FAMILY_CODE
FROM
    ARTICLE_COLOR_LOOKUP ACL
WHERE
    ACL.ARTICLE_CODE NOT IN (SELECT
                                A.ARTICLE_CODE
                                FROM
                                    ARTICLE_LOOKUP A);
-- Table de rejet des clés primaires non uniques de la table ARTICLE_LOOKUP
INSERT INTO [dbo].[ARTICLE_LOOKUP_TRASH]( (
    ARTICLE_CODE
,ARTICLE_LABEL
,CATEGORY
,SALE_PRICE
,FAMILY_NAME
,FAMILY_CODE)
SELECT
    A.ARTICLE_CODE
, A.ARTICLE_LABEL
, A.CATEGORY
, A.SALE_PRICE
, A.FAMILY_NAME
, A.FAMILY_CODE
FROM
    ARTICLE_LOOKUP A
WHERE
    A.ARTICLE_CODE IN ( SELECT
```

```
        ART.ARTICLE_CODE
FROM
        ARTICLE_LOOKUP ART
HAVING
        COUNT(ART.ARTICLE_CODE) > 1
GROUP BY

        ART.ARTICLE_CODE);

-- Table de rejet des clés primaires non uniques de la table CALENDAR_YEAR_LOOKUP
INSERT INTO [dbo].[CALENDAR_YEAR_LOOKUP_TRASH](
    WEEK_KEY

    ,WEEK_IN_YEAR
    ,YEAR
    ,FISCAL_PERIOD
    ,YEAR_WEEK
    ,QUARTER
    ,MONTH_NAME
    ,MONTH
    ,HOLIDAY_FLAG)
SELECT
    C.WEEK_KEY
    ,C.WEEK_IN_YEAR
    ,C.YEAR
    ,C.FISCAL_PERIOD
    ,C.YEAR_WEEK
    ,C.QUARTER
    ,C.MONTH_NAME
    ,C.MONTH
    ,C.HOLIDAY_FLAG
FROM
    CALENDAR_YEAR_LOOKUP C
WHERE
```

```
C.WEEK_KEY IN (SELECT
CAL.WEEK_KEY
FROM
    CALENDAR_YEAR_LOOKUP CAL
HAVING
    COUNT(CAL.WEEK_KEY) > 1
GROUP BY
CAL.WEEK_KEY);
-- Table de rejet des clés primaires non uniques de la table OUTLET_LOOKUP
INSERT INTO [dbo].[OUTLET_LOOKUP_TRASH] (

    SHOP_NAME
    ,ADDRESS_1
    ,MANAGER
    ,DATE_OPEN
    ,OPEN
    ,OWNED_OUTRIGHT
    ,FLOOR_SPACE
    ,ZIP_CODE
    ,CITY
    ,STATE
    ,SHOP_CODE)
SELECT
    O.SHOP_NAME
    ,O.ADDRESS_1
    ,O.MANAGER
    ,O.DATE_OPEN
    ,O.OPEN
    ,O.OWNED_OUTRIGHT
    ,O.FLOOR_SPACE
    ,O.ZIP_CODE
    ,O.CITY
```

```
,O.STATE
,O.SHOP_CODE
FROM
    OUTLET_LOOKUP O
WHERE
    O.SHOP_CODE IN (SELECT
                        OUTLET.SHOP_CODE
                    FROM
                        OUTLET_LOOKUP OUTLET
                    HAVING
                        COUNT(OUTLET.SHOP_CODE) > 1
                    GROUP BY
                        OUTLET.SHOP_CODE);
-- Table de rejet des clés primaires non uniques de la table SHOP_FACTS
INSERT INTO [dbo].[SHOP_FACTS_TRASH] (
    ID
    ,ARTICLE_CODE
    ,COLOR_CODE
    ,WEEK_KEY
    ,SHOP_CODE
    ,MARGIN
    ,AMOUNT_SOLD)
SELECT
    S.ID
    ,S.ARTICLE_CODE
    ,S.COLOR_CODE
    ,S.WEEK_KEY
    ,S.SHOP_CODE
    ,S.MARGIN
    ,S.AMOUNT_SOLD
FROM
```

```
SHOP_FACTS S
WHERE
    S.ID IN (SELECT
                SH.ID
            FROM
                SHOP_FACTS SH
            HAVING
                COUNT(SH.ID) > 1
            GROUP BY
                SH.ID);

-- Tables de rejet des clés étrangères manquantes de la table SHOP_FACTS
-- Table de rejet de FK_WEEK_KEY
INSERT INTO R_SHOP_FACTS_FK_WEEK_KEY(
    ID
    ,ARTICLE_CODE
    ,COLOR_CODE
    ,WEEK_KEY
    ,SHOP_CODE
    ,MARGIN
    ,AMOUNT_SOLD)

SELECT
    S.ID
    ,S.ARTICLE_CODE
    ,S.COLOR_CODE
    ,S.WEEK_KEY
    ,S.SHOP_CODE
    ,S.MARGIN
    ,S.AMOUNT_SOLD
FROM
    SHOP_FACTS S
WHERE
```

```
S.WEEK_KEY NOT IN (SELECT
DISTINCT C.WEEK_KEY
FROM
CALENDAR_YEAR_LOOKUP C);
-- Table de rejet de FK_COLOR_CODE
INSERT INTO R_SHOP_FACTS_FK_COLOR_CODE(
ID
,ARTICLE_CODE
,COLOR_CODE
,WEEK_KEY
,SHOP_CODE
,MARGIN
,AMOUNT_SOLD)
SELECT
S.ID
,S.ARTICLE_CODE
,S.COLOR_CODE
,S.WEEK_KEY
,S.SHOP_CODE
,S.MARGIN
,S.AMOUNT_SOLD
FROM
SHOP_FACTS S
WHERE
S.COLOR_CODE NOT IN ( SELECT
DISTINCT ACL.COLOR_CODE
FROM
ARTICLE_COLOR_LOOKUP ACL);
-- Table de rejet de FK_ARTICLE_CODE
INSERT INTO R_SHOP_FACTS_FK_COLOR_CODE(
ID
```

```
,ARTICLE_CODE
,COLOR_CODE
,WEEK_KEY
,SHOP_CODE
,MARGIN
,AMOUNT_SOLD)
SELECT
    S.ID
    ,S.ARTICLE_CODE
    ,S.COLOR_CODE
    ,S.WEEK_KEY
    ,S.SHOP_CODE
    ,S.MARGIN
    ,S.AMOUNT_SOLD
FROM
    SHOP_FACTS S
WHERE
    S.ARTICLE_CODE NOT IN ( SELECT
                                DISTINCT A.ARTICLE_CODE
                                FROM
                                    ARTICLE_LOOKUP A);
COMMIT;
```


c) Data quality

FK ARTICLE_COLOR_LOOKUP

```
SELECT
    ACL.ARTICLE_CODE
FROM
    ARTICLE_COLOR_LOOKUP ACL
WHERE
    ACL.ARTICLE_CODE NOT IN
        (SELECT
            ALO.ARTICLE_CODE
        FROM
            ARTICLE_LOOKUP ALO)
ORDER BY
    ACL.ARTICLE_CODE ASC;
```

FK SHOP FACTS

```
--***** WEEK_KEY *****-----  
  
SELECT  
    SF.ID "ID SHOP FACTS"  
    ,SF.WEEK_KEY  
  
FROM  
    SHOP_FACTS SF  
  
WHERE  
    SF.WEEK_KEY NOT IN  
        (SELECT  
            DISTINCT CYL.WEEK_KEY  
        FROM  
            CALENDAR_YEAR_LOOKUP CYL)  
  
ORDER BY  
    SF.ID ASC;
```

--***** ARTICLE_CODE *****-----

SELECT

SF.ID

,SF.ARTICLE_CODE

FROM

SHOP_FACTS SF

WHERE

SF.ARTICLE_CODE NOT IN

(SELECT

DISTINCT AL.ARTICLE_CODE

FROM

ARTICLE_LOOKUP AL)

ORDER BY

SF.ID ASC;

--***** COLOR_CODE *****-----

SELECT

SF.ID

, SF.COLOR_CODE

FROM

SHOP_FACTS SF

WHERE

SF.COLOR_CODE NOT IN

(SELECT

DISTINCT ACL.COLOR_CODE

FROM

ARTICLE_COLOR_LOOKUP ACL)

ORDER BY

SF.ID ASC;

PK ARTICLE COLOR LOOKUP

```
--***** ARTICLE_CODE, COLOR_CODE *****-----  
-----  
  
SELECT  
  
    ACL.ARTICLE_CODE  
  
    , ACL.COLOR_CODE  
  
    , COUNT(CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)) AS  
"NB_PK_ARTICLE_CODE_COLOR_CODE"  
  
FROM  
  
    ARTICLE_COLOR_LOOKUP ACL  
  
GROUP BY  
  
    ACL.ARTICLE_CODE  
  
    , ACL.COLOR_CODE  
  
    , CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)  
  
HAVING  
  
    COUNT(CONCAT(ACL.ARTICLE_CODE,ACL.COLOR_CODE)) > 1
```

PK ARTICLE LOOKUP

```
--***** ARTICLE_CODE *****-----  
  
SELECT  
  
    A.ARTICLE_CODE  
    , COUNT(A.ARTICLE_CODE) AS "NB_PK_ARTICLE_CODE"  
  
FROM  
  
    ARTICLE_LOOKUP A  
  
    HAVING  
  
        COUNT(A.ARTICLE_CODE) > 1  
  
  
GROUP BY  
  
    A.ARTICLE_CODE  
  
ORDER BY  
  
    A.ARTICLE_CODE ASC;
```

PK CALENDAR YEAR LOOKUP

```
--***** WEEK_KEY *****-----  
  
SELECT  
  
    C.WEEK_KEY  
    , COUNT(C.WEEK_KEY) AS "NB_PK_WEEK_KEY"  
  
FROM  
  
    CALENDAR_YEAR_LOOKUP C  
  
    HAVING  
  
        COUNT(C.WEEK_KEY) > 1  
  
GROUP BY  
  
    C.WEEK_KEY;
```

PK OUTLET LOOKUP

```
--***** SHOP_CODE *****-----  
  
SELECT  
  
    O.SHOP_CODE  
    , COUNT(O.SHOP_CODE) AS "NB_PK_SHOP_CODE"  
  
FROM  
  
    OUTLET_LOOKUP O  
  
    HAVING  
  
        COUNT(O.SHOP_CODE) > 1  
  
GROUP BY  
  
    O.SHOP_CODE;
```

PK SHOP FACTS

```
--***** ID *****  
  
SELECT  
    *  
  
FROM  
    SHOP_FACTS  
  
WHERE  
    ID IN  
    (  
        SELECT ID  
        FROM SHOP_FACTS  
        GROUP BY ID  
        HAVING COUNT(*)>1  
    )  
  
ORDER BY  
    ID;  
  
COMMIT;
```

2. Partitionnement

SHOP_FACTS

