

BUSINESS INTELLIGENCE & DATA WAREHOUSE

Développement d'un système de Business Intelligence pour la gestion des ventes de magasins avec la base de données EMODE



Table des matières

Table des matières	1
Introduction	3
Partie 1 : Implémentation des fonctions ETL	4
Schéma de la base de données	4
La qualité des données	4
Les clés primaires et étrangères	4
Vérification des clés primaires	5
Vérification des clés étrangères	6
Les tables de rejet	7
Les packages SQL Server Integration Services (SSIS)	8
Premier package	8
La suppression des données dans la base de données de destination	9
Le transfert des données des tables de dimension	10
Le transfert des données de la table de faits	10
Deuxième package	12
Transfert incrémental	13
Traçabilité du transfert	16
Troisième package	17
Quatrième package	18
Automatisation du processus	20
Exécution immédiate des packages	20
Planification d'une tâche au niveau du système d'exploitation	21
Utilisation des "jobs" de SQL Server Agent	23
Le test de l'ETL	23
Le transfert initial	24
Le transfert incrémental	24
Partie 2 : Optimisation de l'entrepôt de données	26
Partitionnement de la table de faits	26
Les groupes de fichiers	26
La fonction et le schéma de partition	27
La création de la table	27
Le projet SQL Server Analysis Services (SSAS)	27
Partie 3 : Implémentation du reporting	28
Le projet SQL Server Reporting Services (SSRS)	28
Création de l'univers Business Objects (BO)	28

Reporting avec Web Intelligence	28
Reporting avec Microsoft Excel	28
Reporting avec Qlikview	28
Conclusion	29

Introduction

L'informatique décisionnelle ou en anglais **Business Intelligence (BI)** est l'informatique à l'usage des décideurs et des dirigeants d'entreprises. Elle permet, au sein d'une entreprise, à un décideur d'avoir une vue d'ensemble de l'activité traitée et ainsi d'offrir une aide à la décision.

Dans le cadre de notre formation au titre d'ingénieur, nous avons été amenés à concevoir et à implémenter un système décisionnel portant sur la **gestion des ventes de magasins autour de la base de données EMODE** fournie.

Notre réalisation a suivi trois grandes étapes :

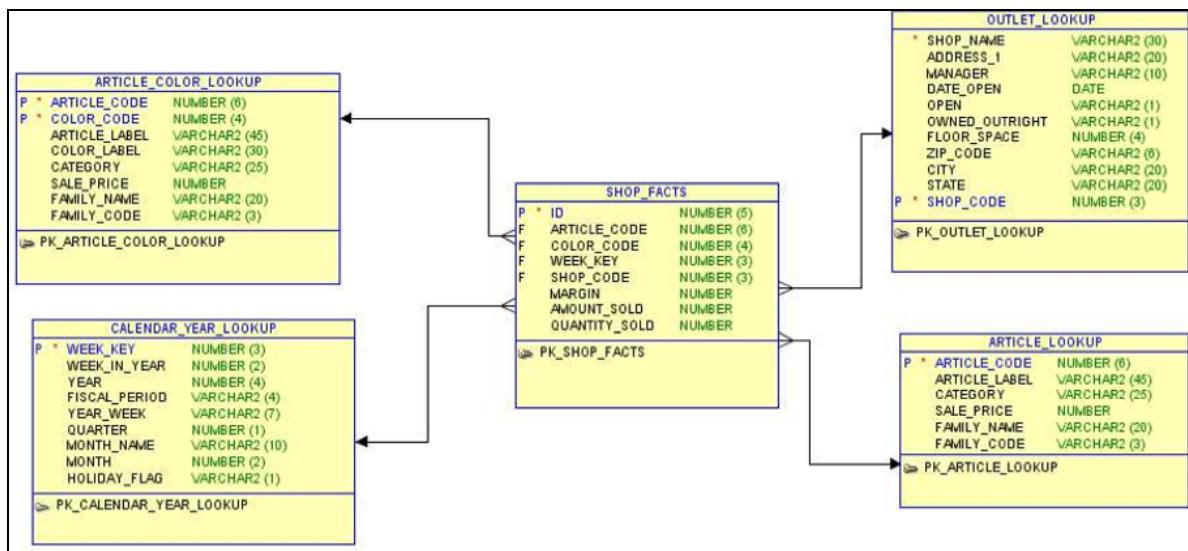
- La première étape a consisté au transfert, grâce aux fonctions **ETL**, des données de la base source sur **Oracle 12C** vers la base de destination sur **SQL Server 2016**.
- La deuxième étape a consisté en l'optimisation de l'entrepôt de données grâce à la mise en place d'un partitionnement.
- La troisième et dernière étape a consisté à créer des rapports (tableaux classiques, tableaux croisés dynamiques et graphiques) avec différents outils.

Le but de ce document est de présenter et d'expliquer en détails les différentes étapes de la réalisation du projet.

Partie 1 : Implémentation des fonctions ETL

1. Schéma de la base de données

Toutes les tables de la base *EMODE* n'étant pas nécessaires à notre projet, il est important de présenter le schéma définitif qui sera utilisé. Seules les tables *SHOP_FACTS*, *OUTLET_LOOKUP*, *ARTICLE_LOOKUP*, *ARTICLE_COLOR_LOOKUP*, *CALENDAR_YEAR_LOOKUP* seront utilisées. On a ainsi un modèle en étoile où *SHOP_FACTS* est la table de faits et les quatre autres tables sont les tables de dimensions.



2. La qualité des données

a. Les clés primaires et étrangères

La base de données source ne présentant aucune contrainte d'intégrité il peut exister des doublons ou des données n'existant dans la table de dimension mais présentes dans la table de fait. Il est donc nécessaire de vérifier la qualité des données avant le transfert afin d'obtenir un jeu de données sain. La vérification est effectuée à l'aide de plusieurs requêtes SQL.

i. Vérification des clés primaires

Une clé primaire identifiant de manière unique une ligne d'une table, la vérification va consister, à l'aide d'une requête SQL, à s'assurer qu'elle a une valeur unique pour chaque ligne de la table.

```
SELECT
    'ARTICLE_LOOKUP' as TAB,
    COUNT(article_code) as PK_LINES,
    COUNT(DISTINCT article_code) as DISTINCK_PK_LINES
FROM article_lookup
WHERE article_code IS NOT NULL
UNION

SELECT
    'ARTICLE_COLOR_LOOKUP' as TAB,
    COUNT(CONCAT(cast(ARTICLE_code as char(6)),cast(COLOR_CODE as char(4)))) as PK_LINES,
    COUNT(DISTINCT CONCAT(to_char(ARTICLE_CODE),to_char(COLOR_CODE)))
FROM ARTICLE_COLOR_LOOKUP
WHERE article_code IS NOT NULL
AND COLOR_CODE IS NOT NULL
UNION

SELECT
    'CALENDAR_YEAR_LOOKUP' as TAB,
    COUNT(WEEK_KEY) as PK_LINES,
    COUNT(DISTINCT WEEK_KEY) as DISTINCK_PK_LINES
FROM CALENDAR_YEAR_LOOKUP
WHERE WEEK_KEY IS NOT NULL
UNION

SELECT
    'OUTLET_LOOKUP' as TAB,
    COUNT(shop_code) as PK_LINES,
    COUNT(DISTINCT SHOP_CODE) as DISTINCK_PK_LINES
FROM OUTLET_LOOKUP
WHERE shop_code IS NOT NULL
UNION

SELECT
    'SHOP_FACTS' as TAB,
    COUNT(ID) as PK_LINES,
    COUNT(DISTINCT ID) as DISTINCK_PK_LINES
FROM SHOP_FACTS
WHERE ID IS NOT NULL;
```

TAB	PK_LINES	DISTINCK_PK_LINES
ARTICLE_COLOR_LOOKUP	663	661
ARTICLE_LOOKUP	211	211
CALENDAR_YEAR_LOOKUP	262	262
OUTLET_LOOKUP	13	13
SHOP_FACTS	89171	89171

Il ressort que la table *ARTICLE_COLOR_LOOKUP* comporte des données en doubles. Ces données peuvent être identifiées par une autre requête SQL.

```
SELECT
    count(*) as nombre_enregistrements,
    count(distinct concat(to_char(article_code),to_char(color_code))) as identifiants,
    article_code,
    color_code
FROM
    ARTICLE_COLOR_LOOKUP
GROUP BY
    color_code, article_code
HAVING
    count(*)>1
;
```

NOMBRE_ENREGISTREMENTS	IDENTIFIANTS	ARTICLE_CODE	COLOR_CODE
2	1	170016	902
2	1	170016	210

Les données incohérentes seront plus tard transférées dans des tables de rejet.

ii. Vérification des clés étrangères

La table *SHOP_FACTS* contient plusieurs références vers des lignes des tables de dimension. Ces références impliquent l'existence des lignes référencées dans les dimensions. Il est donc nécessaire de vérifier que les clés étrangères présentes dans la table de faits existent comme clé primaire dans les tables de dimension.

```

SELECT
    COUNT( DISTINCT CONCAT(to_char(article_code),to_char(color_code))) AS "Occurrences"
    ,article_code
    ,color_code
FROM
    SHOP_FACTS
WHERE
    CONCAT(to_char(article_code),to_char(color_code)) not in
(
    SELECT
        CONCAT(to_char(article_code),to_char(color_code))
    FROM
        ARTICLE_COLOR_LOOKUP
)
GROUP BY
    article_code
    ,color_code
;

```

Occurrences	ARTICLE_CODE	COLOR_CODE
1	177264	627
1	177264	182
1	177264	199
1	177264	1103
1	177264	1224
1	177264	308
1	177264	138
1	177264	901
1	177264	423
1	177264	210
1	177264	731
1	177264	1228
1	177264	902
1	177264	612

De cette vérification nous avons tiré qu'au sein de la table *SHOP_FACTS* il y a des lignes possédant des références à des lignes de la table *ARTICLE_COLOR_LOOKUP* qui sont invalides. Ces données seront aussi transférées la table de rejet correspondante.

b. Les tables de rejet

Certaines données ne respectant pas les contraintes de clés primaires ou étrangères, ont été créées des tables de rejet pour chaque table afin de conserver ces données. Ces tables de rejet reprennent exactement les mêmes structures que nos tables d'origine.

```
SELECT
    AL.ARTICLE_CODE
    , AL.ARTICLE_LABEL
    , AL.CATEGORY
    , AL.SALE_PRICE
    , AL.FAMILY_NAME
    , AL.FAMILY_CODE INTO ARTICLE_LOOKUP_REJECT
FROM
    ARTICLE_LOOKUP AL
WHERE
    AL.ARTICLE_CODE IN (
        SELECT
            AL2.ARTICLE_CODE
        FROM
            ARTICLE_LOOKUP AL2
        GROUP BY
            AL2.ARTICLE_CODE
        HAVING
            COUNT(AL2.ARTICLE_CODE)>1);
```

3. Les packages SQL Server Integration Services (SSIS)

a. Premier package

Le but de ce package est de transférer la totalité des données de la base *EMODE* d'Oracle vers celle de SQL Server qui sera utilisée pour le reporting. Cela passe par :

- la suppression des données dans les tables de la base de destination,
- le transfert des données cohérentes des tables sources vers les tables de destination correspondantes,
- le transfert des données incohérentes des tables sources dans les tables de rejet.

L'existence de contraintes de clés étrangères empêchant l'utilisation de l'instruction *TRUNCATE* pour la suppression des données, il a fallu les désactiver avant la suppression des données puis les réactiver après. Ces mêmes contraintes imposent que le transfert des données se fasse dans un ordre précis : les données dans les tables statiques avant celles des tables dynamiques (table contenant des clés étrangères).

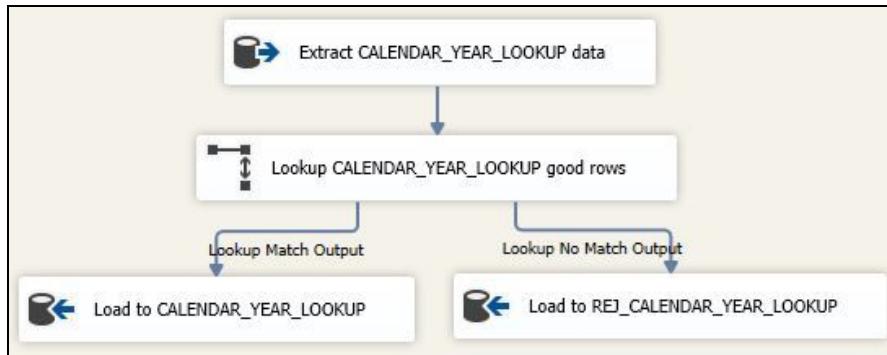


i. La suppression des données dans la base de données de destination

La suppression des données dans la base se fait donc avec l'instruction *TRUNCATE* car ne déclenchant pas de trigger et étant moins gourmande en ressource que *DELETE*. De plus cette instruction réinitialise les valeurs de colonnes d'entiers avec la propriété d'auto-incrémentation.

ii. Le transfert des données des tables de dimension

Pour chaque table de dimension, nous récupérons toutes les données de la base de données *EMODE* d'Oracle puis nous testons l'intégrité des données avant de les transférer vers deux tables différentes en fonction des situations.



La vérification de l'intégrité des données se fait à l'aide d'un *LOOKUP*. Les données cohérentes (pas de doublons, contraintes d'intégrité référentielle respectées) sont transférées dans les bonnes tables. Dans le cas contraire ces données sont redirigées vers les tables de rejet créées au préalable avec la même structure que les bonnes tables.

```

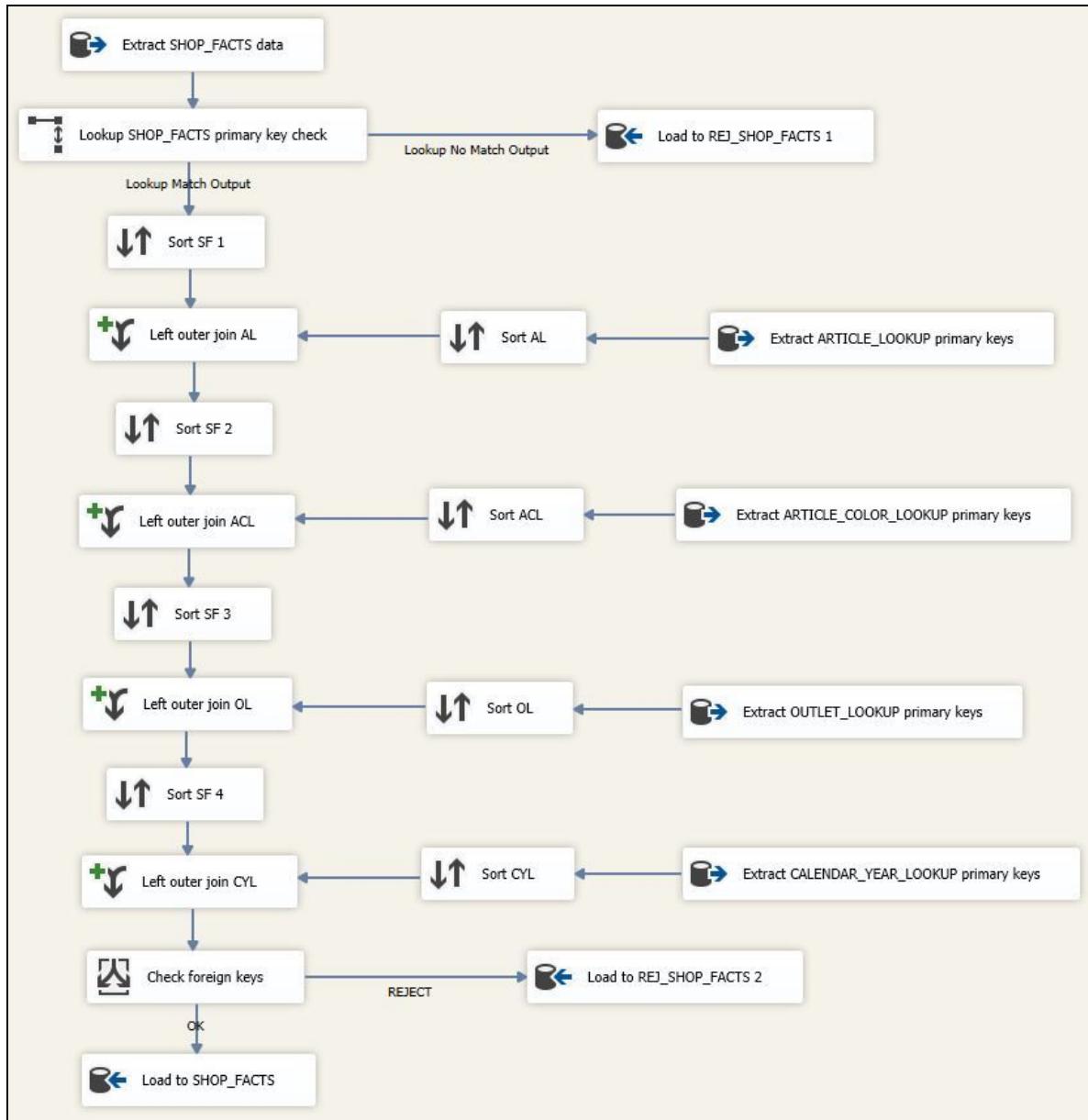
SELECT * INTO REJ_ARTICLE_LOOKUP FROM ARTICLE_LOOKUP WHERE 1=2;
SELECT * INTO REJ_ARTICLE_COLOR_LOOKUP FROM ARTICLE_COLOR_LOOKUP WHERE 1=2;
SELECT * INTO REJ_CALENDAR_YEAR_LOOKUP FROM CALENDAR_YEAR_LOOKUP WHERE 1=2;
SELECT * INTO REJ_OUTLET_LOOKUP FROM OUTLET_LOOKUP WHERE 1=2;
SELECT * INTO REJ_SHOP_FACTS FROM SHOP_FACTS WHERE 1=2;
  
```

iii. Le transfert des données de la table de faits

Pour la table des faits, il faut s'assurer que les différentes clés étrangères sont présentes dans les différentes tables de dimension. Pour ce faire nous prendrons comme référence les données de la base de destination, l'intégrité des des données de la base source puisque n'étant pas assurée.

La transformation *LOOKUP* sert à faire une équijointure entre des données issues d'une même source et ne peut donc être utilisée dans le cas de notre table de faits. Pour comparer des données issues de sources différentes, il existe la transformation SSIS

Nous utiliserons donc *MERGE JOIN* qui nous permettra une jointure entre des tables issues de la base source et des tables de la base de destination.



Pour chacune des jointures externes entre les clés étrangères de la table de faits de la source et les clés primaires des tables de dimension de la destination, nous ajoutons une colonne avec la clé primaire issue de la table de dimension. Un test sera ainsi effectué sur ces clés primaires à la fin du Data Flow ; si un des champs est vide cela signifie qu'il n'y a pas de référence dans la table de dimension, la ligne peut donc être rejetée.

b. Deuxième package

Après le premier transfert de données, les premières données étant présentes dans la base de destination, il peut être fastidieux et coûteux en ressources d'effectuer un transfert complet à nouveau lorsqu'il y a mise à jour des données dans la base source. Ce deuxième package viendra en support du premier afin de ne transférer que les modifications lors de la mise à jour des données dans la base de données source. On parle de transfert incrémental. De plus ce package intégrera la traçabilité du transfert avec une table d'audit créée en SQL.

Ce package se déroule comme suit : Les données sont récupérées d'une base incrémentale qui contient les modifications opérées depuis le dernier transfert. Ensuite est effectuée une vérification sur les contraintes puis, en fonction des résultats, les données sont redirigées dans les tables spécifiques. Au fur et à mesure que ces opérations s'exécutent, nos tables d'audit sont peuplées. A la fin du transfert, les données des tables de la base incrémentale sont supprimées.



i. Transfert incrémental

Afin d'effectuer le transfert incrémental, il est nécessaire de créer un nouveau schéma qui contiendra uniquement les dernières modifications apportées au schéma *EMODE* de base sous Oracle. Ce schéma portera le nom de *EMODE_INC*.

```

Copyright (c) 1982, 2014, Oracle. All rights reserved.
SQL> connect / as sysdba
Connecté.
SQL> create user emode_inc identified by emode_inc;
Utilisateur créé.
SQL> grant connect, resource to emode_inc;
Autorisation de priviléges (GRANT) acceptée.
SQL> commit;
Validation effectuée.

```

EMODE et *EMODE_INC* seront donc semblable à une différence près ; dans chaque table de *EMODE_INC*, nous rajouterons une colonne *OP_TYPE* renseignant le type d'opération de mise à jour : 'i' pour insertion, 'd' pour suppression, et 'u' pour modification.

```
CREATE TABLE ARTICLE_LOOKUP (
    ARTICLE_CODE NUMBER(6,0),
    ARTICLE_LABEL VARCHAR2(45),
    CATEGORY VARCHAR2(25),
    SALE_PRICE NUMBER(13,2),
    FAMILY_NAME VARCHAR2(20),
    FAMILY_CODE VARCHAR2(3),
    OP_TYPE CHAR(1),
    CONSTRAINT CHECK_OP_TYPE_AL CHECK (OP_TYPE IN('i','u','d'))
);
```

Afin que nos triggers depuis *EMODE* puissent écrire dans *EMODE_INC*, il a été nécessaire d'ajouter certains droits.

```
GRANT INSERT, UPDATE, DELETE, SELECT
ON ARTICLE_COLOR_LOOKUP
TO emode;

GRANT INSERT, UPDATE, DELETE, SELECT
ON ARTICLE_LOOKUP
TO emode;

GRANT INSERT, UPDATE, DELETE, SELECT
ON OUTLET_LOOKUP
TO emode;

GRANT INSERT, UPDATE, DELETE, SELECT
ON CALENDAR_YEAR_LOOKUP
TO emode;

GRANT INSERT, UPDATE, DELETE, SELECT
ON SHOP_FACTS
TO emode;
```

L'alimentation en données de *EMODE_INC* se fera grâce à des triggers mis en place au niveau de *EMODE*. Ces derniers seront automatiquement déclenchés lorsqu'une modification sera effectuée sur *EMODE*. Il y aura pour chaque table un trigger qui traitera les différents types de modifications possibles : insertion, mise à jour et suppression. Afin de faciliter l'appel aux nouvelles tables créées nous avons créé des synonymes ou alias.

```
CREATE SYNONYM INC_ARTICLE_COLOR_LOOKUP FOR EMODE_INC.ARTICLE_COLOR_LOOKUP;
CREATE SYNONYM INC_ARTICLE_LOOKUP FOR EMODE_INC.ARTICLE_LOOKUP;
CREATE SYNONYM INC_CALENDAR_YEAR_LOOKUP FOR EMODE_INC.CALENDAR_YEAR_LOOKUP;
CREATE SYNONYM INC_OUTLET_LOOKUP FOR EMODE_INC.OUTLET_LOOKUP;
CREATE SYNONYM INC_SHOP_FACTS FOR EMODE_INC.SHOP_FACTS;
```

```
-- ARTICLE_LOOKUP
-- Insert
CREATE OR REPLACE TRIGGER TRG_INS_ARTICLE_LOOKUP AFTER INSERT ON ARTICLE_LOOKUP FOR EACH ROW
BEGIN
INSERT INTO INC_ARTICLE_LOOKUP VALUES(:NEW.ARTICLE_CODE, :NEW.ARTICLE_LABEL, :NEW.CATEGORY, :NEW.SALE_PRICE, :NEW.FAMILY_NAME, :NEW.FAMILY_CODE, 'i');
END;
-- Update
CREATE OR REPLACE TRIGGER TRG_UPD_ARTICLE_LOOKUP AFTER UPDATE ON ARTICLE_LOOKUP FOR EACH ROW
BEGIN
INSERT INTO INC_ARTICLE_LOOKUP VALUES(:NEW.ARTICLE_CODE, :NEW.ARTICLE_LABEL, :NEW.CATEGORY, :NEW.SALE_PRICE, :NEW.FAMILY_NAME, :NEW.FAMILY_CODE, 'u');
END;
-- Delete
CREATE OR REPLACE TRIGGER TRG_DEL_ARTICLE_LOOKUP AFTER DELETE ON ARTICLE_LOOKUP FOR EACH ROW
BEGIN
INSERT INTO INC_ARTICLE_LOOKUP VALUES(:OLD.ARTICLE_CODE, :OLD.ARTICLE_LABEL, :OLD.CATEGORY, :OLD.SALE_PRICE, :OLD.FAMILY_NAME, :OLD.FAMILY_CODE, 'd');
END;
```

ii. Traçabilité du transfert

Lors de l'exécution automatique d'un transfert, il est utile et intéressant d'avoir des informations pour savoir si le transfert s'est bien déroulé mais aussi pour avoir des statistiques sur les données transférées ou pour récupérer les erreurs survenues.

Ainsi dans le but d'auditer ce package nous avons mis en place trois tables ayant chacune une utilité particulière :

- *AUDIT_TRACE* qui va permettre de récolter des informations d'ordre général comme la date et heure du début et de fin du transfert et un numéro unique pour identifier le transfert.

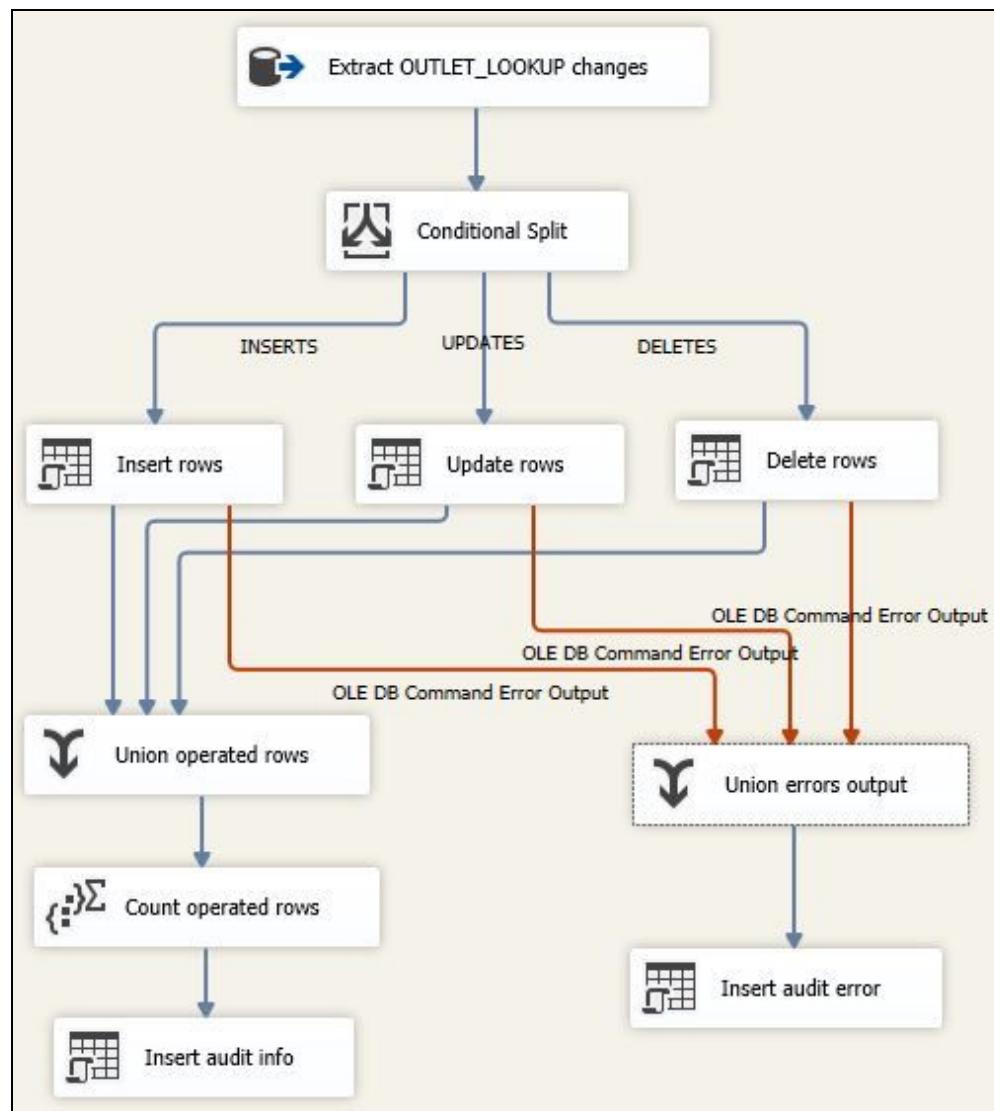
```
CREATE TABLE AUDIT_TRACE (
    TRANSFER_NUM int identity(1,1) NOT NULL,
    TRANSFER_BEGIN_TIME datetime NOT NULL,
    TRANSFER_END_TIME datetime NULL,
    CONSTRAINT PK_AUDIT_TRACE PRIMARY KEY (TRANSFER_NUM)
);
```

- *AUDIT_INFO* qui servira à enregistrer les informations de transferts effectués comme le nombre de lignes traitées et l'opération effectuée.

```
CREATE TABLE AUDIT_INFO (
    INFO_NUM int identity(1,1) NOT NULL,
    TRANSFER_NUM int NOT NULL,
    TABLE_NAME varchar(20) NOT NULL,
    ROWS_AFFECTED int NOT NULL,
    OP_TYPE char(1) NOT NULL,
    CONSTRAINT PK_AUDIT_INFO PRIMARY KEY (INFO_NUM),
    CONSTRAINT FK_AUDIT_INFO_TRANSFER_NUM FOREIGN KEY (TRANSFER_NUM) REFERENCES AUDIT_TRACE(TRANSFER_NUM),
    CONSTRAINT CHECK_OP_TYPE_AI CHECK (OP_TYPE IN('i','d','u'))
);
```

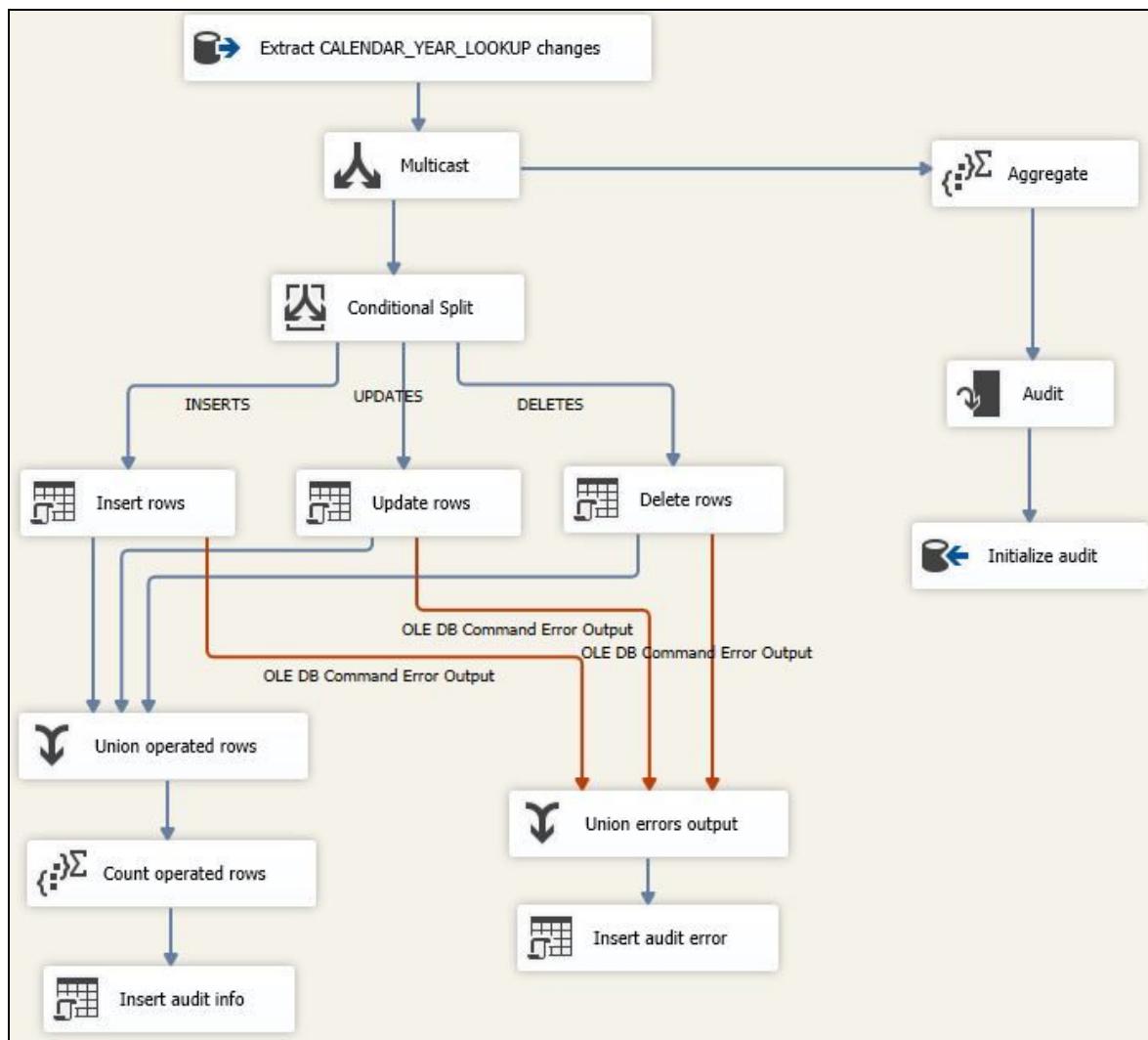
- AUDIT_ERROR qui a pour but de récolter les informations lors d'erreurs durant le transfert et d'indiquer la source du problème.

```
CREATE TABLE AUDIT_ERROR (
    ERROR_NUM int identity(1,1) NOT NULL,
    TRANSFER_NUM int NOT NULL,
    TABLE_NAME varchar(20) NOT NULL,
    PK_VALUE varchar(20),
    OP_TYPE char(1) NOT NULL,
    ERROR_CODE varchar(25),
    CONSTRAINT PK_AUDIT_ERROR PRIMARY KEY (ERROR_NUM),
    CONSTRAINT FK_AUDIT_ERROR_TRANSFER_NUM FOREIGN KEY (TRANSFER_NUM) REFERENCES AUDIT_TRACE(TRANSFER_NUM),
    CONSTRAINT CHECK_OP_TYPE_AE CHECK (OP_TYPE IN('i','d','u'))
);
```



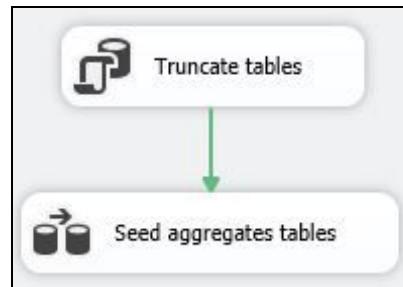
c. Troisième package

Un audit manuel a été mis en place dans le package précédent. Il existe néanmoins des mécanismes d'audit automatique proposés par SSIS. Le package 3 consistera donc à utiliser ces mécanismes. Ce dernier présentera un déroulement similaire au précédent. La différence réside dans l'audit. Nous utiliserons la transformation *AUDIT* qui nous permet d'obtenir des informations d'audit directement de SSIS.



d. Quatrième package

Ce package permettra, après chaque transfert de la base source vers la base de destination, la mise à jour de tables d'agrégats. Cela passe par la suppression des anciennes données des tables concernées puis l'insertion en se basant sur les nouvelles données.



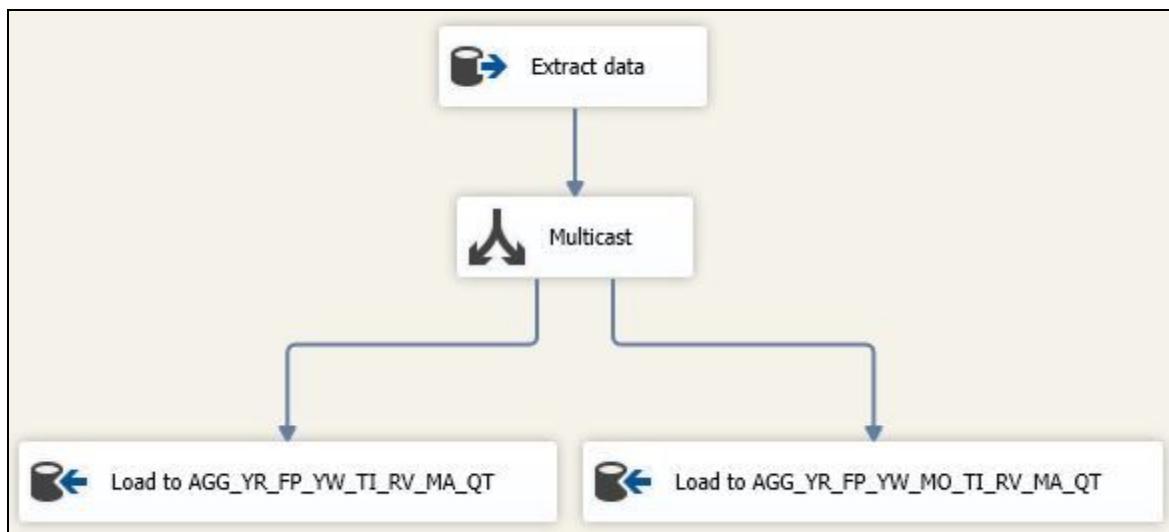
Les tables d'agrégats ont bien-sûr été créées préalablement.

```
CREATE TABLE AGG_YR_FP_YW_MO_TI_RV_MA_QT (
    WEEK_KEY numeric(3, 0) NOT NULL,
    YEAR numeric(4, 0) NULL,
    FISC varchar(4) NULL,
    YEAR_WE varchar(7) NULL,
    MONTH numeric(2, 0) NULL,
    MONTH_NAME varchar(10) NULL,
    TICKETS int NULL,
    RV numeric(38, 2) NULL,
    MARGE numeric(38, 2) NULL,
    QTE numeric(38, 2) NULL
);

CREATE TABLE AGG_YR_FP_YW_TI_RV_MA_QT (
    WEEK_KEY numeric(3, 0) NOT NULL,
    YEAR numeric(4, 0) NULL,
    FISC varchar(4) NULL,
    YEAR_WE varchar(7) NULL,
    TICKETS int NULL,
    RV numeric(38, 2) NULL,
    MARGE numeric(38, 2) NULL,
    QTE numeric(38, 2) NULL
);
```

La mise à jour des tables d'agrégats se fait comme suit :

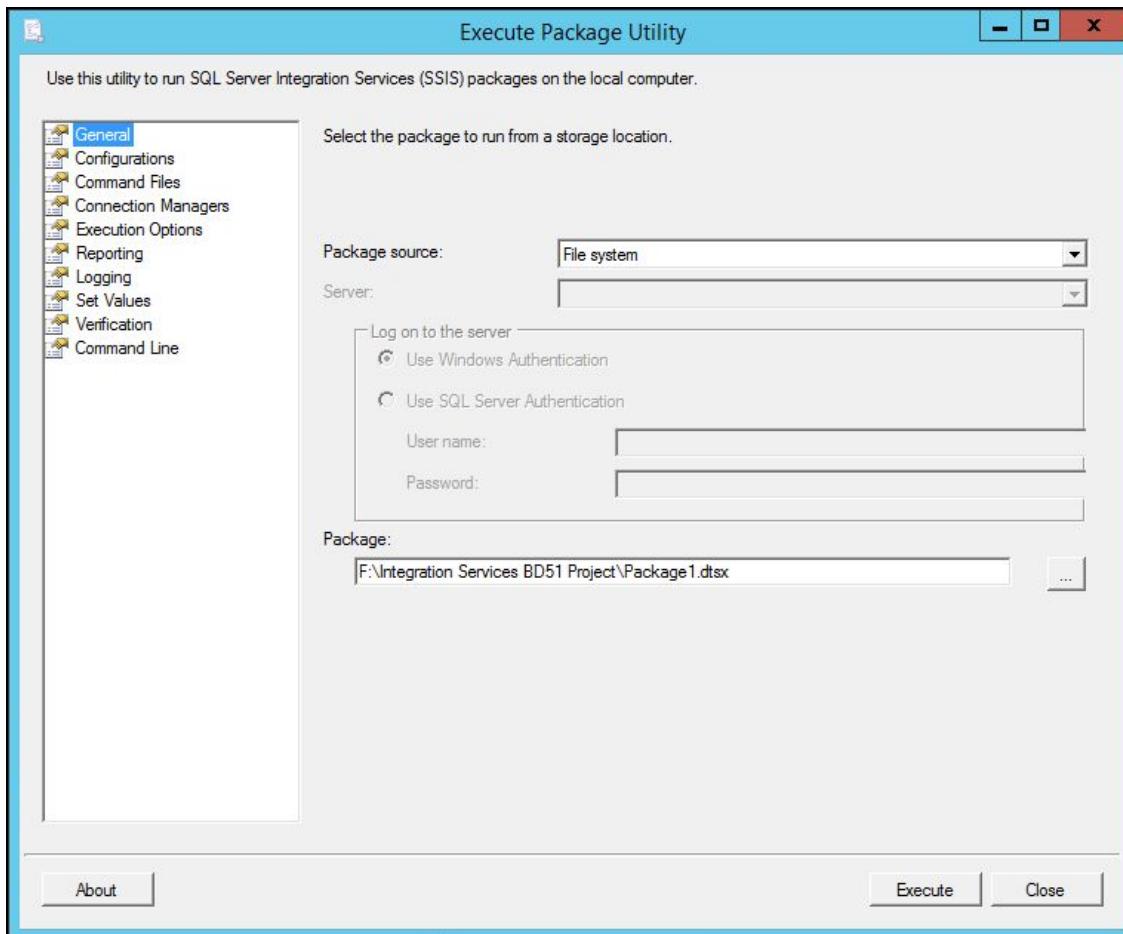
- On sélectionne les données nécessaires pour alimenter les deux tables d'agrégations
- On utilise un *MULTICAST* afin d'effectuer des traitements en parallèle des données sélectionnées
- Les données agrégées sont ensuite triées pour être insérées dans les tables d'agrégats



e. Automatisation du processus

i. Exécution immédiate des packages

Dans les livrables se trouvent, dans le répertoire "ETL" les fichiers de packages qui peuvent être reconnus à leur extension ".dtsx". Il suffit d'un double-clic sur le fichier de package à exécuter pour le lancer.

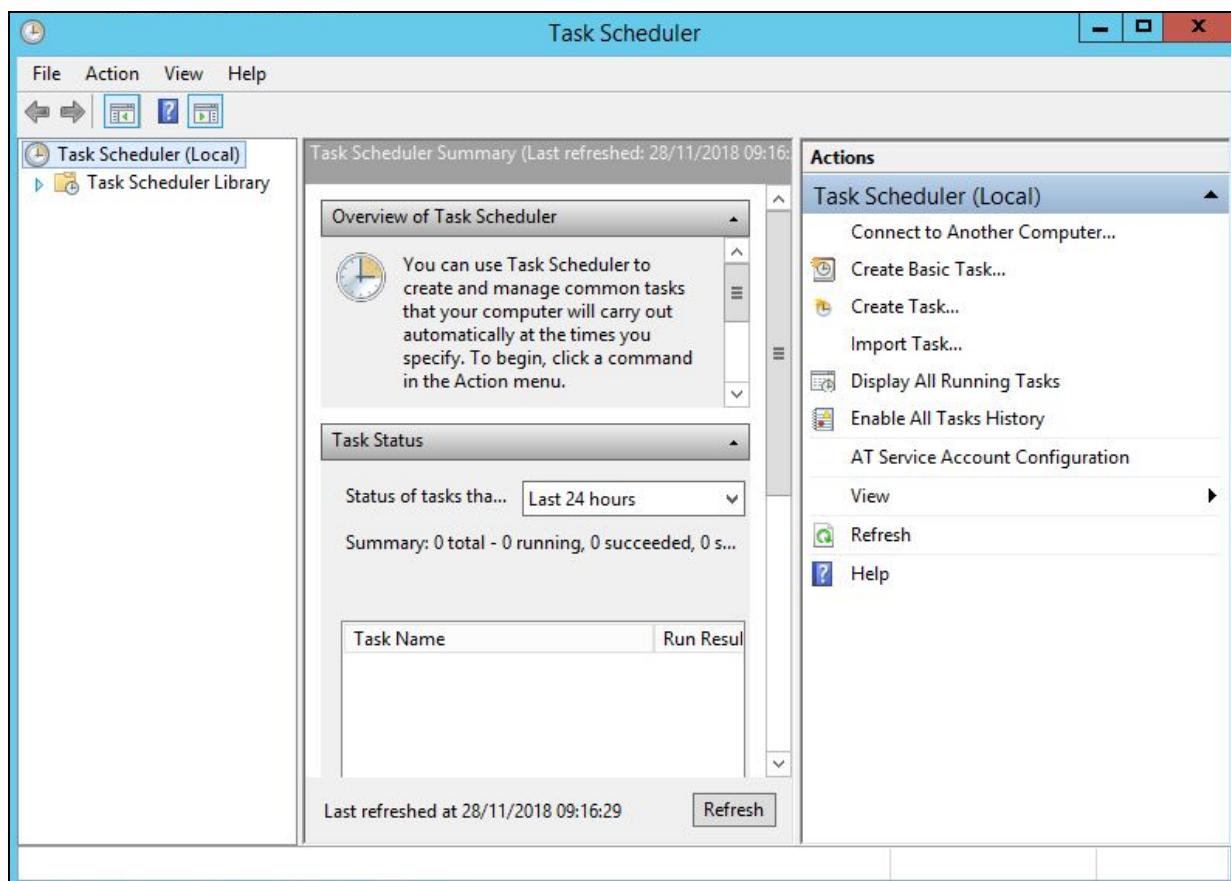


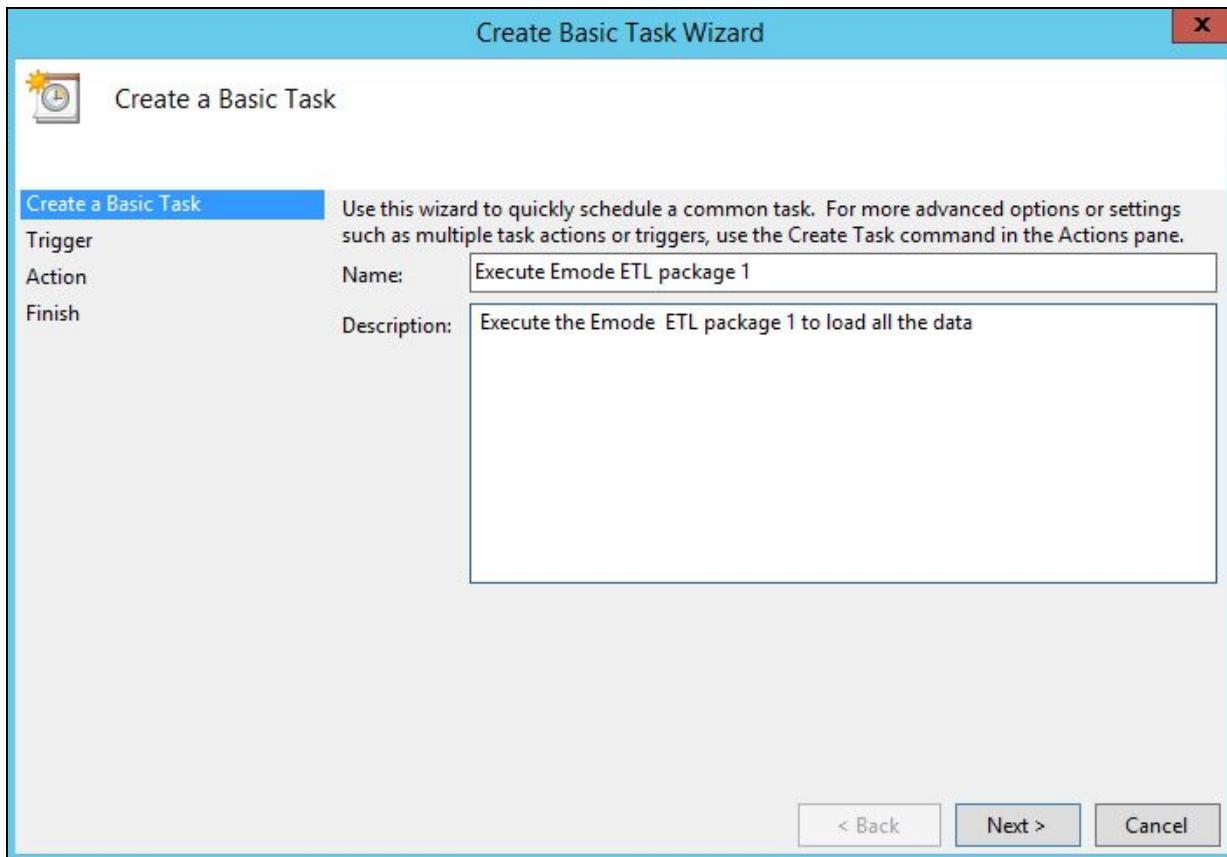
Il est aussi possible d'utiliser un outil en ligne de commandes pour lancer l'exécution d'un package : dtexec. A cet effet nous avons écrit un fichier batch permettant de lancer l'exécution d'un package.

```
@echo off
"C:\Program Files (x86)\Microsoft SQL Server\140\DTS\Binn\DTExec.exe" /f "E:\scripts\Integration Services BD51 Project\Package1.dtsx" > exec_package1.log
echo EXECUTING Package1
echo Done. You can check the logs file
echo.
pause
```

ii. Planification d'une tâche au niveau du système d'exploitation

Grâce à l'outil "Task Scheduler" il est possible de planifier l'exécution d'instructions sur notre système d'exploitation. Il va ainsi nous servir à lancer de manière planifiée les packages. Pour ce faire il faut se rendre dans le "Panneau de Configuration" et rechercher "Tâches planifiées". Il faut donc créer de nouvelles tâches pour les packages que nous voulons exécuter. Pour chaque tâche il faudra renseigner des informations de base comme le nom et la description, configurer le déclencheur et enfin choisir l'action à exécuter (dans notre cas lancer le fichier batch que nous avons créé plus tôt).

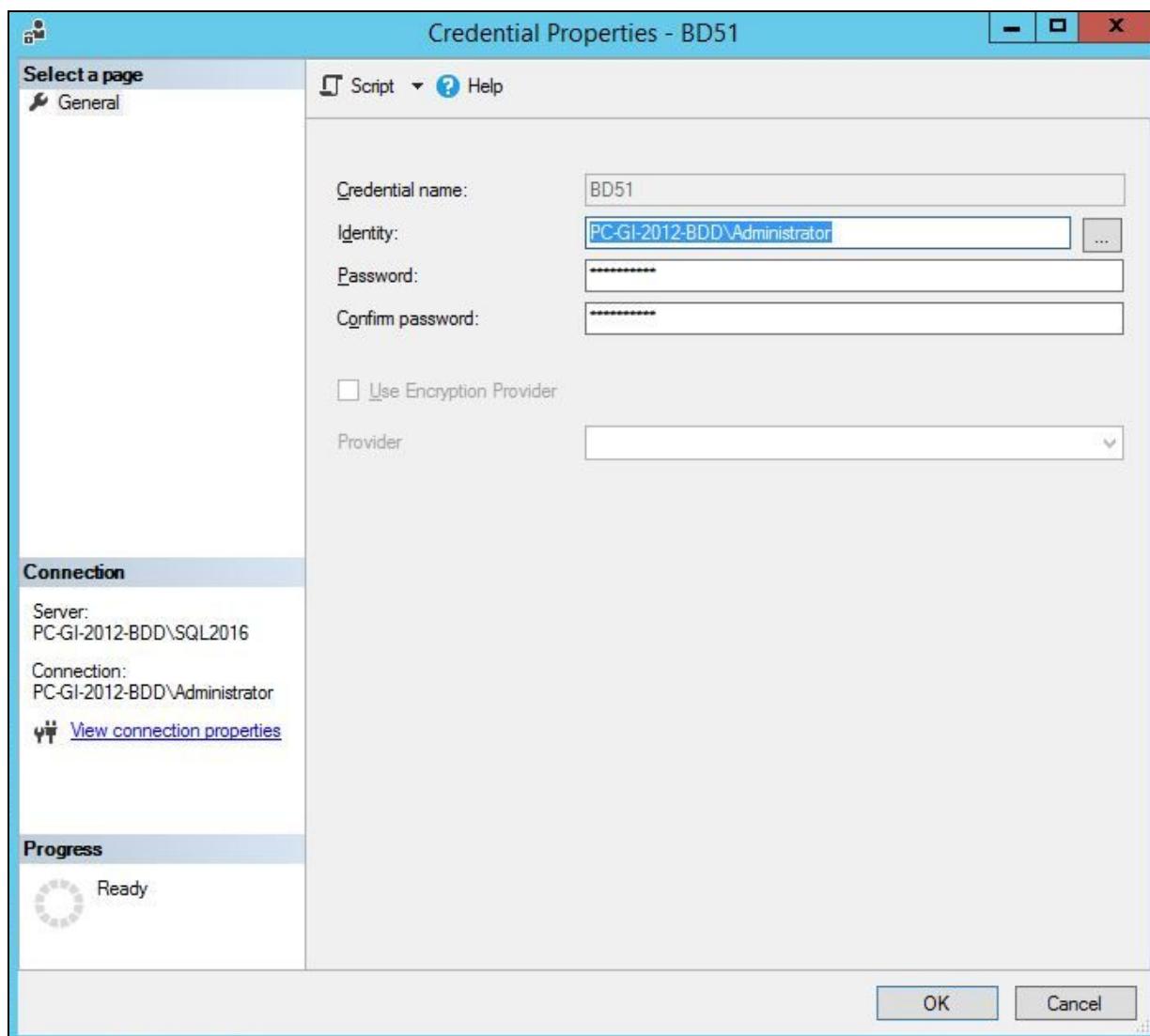


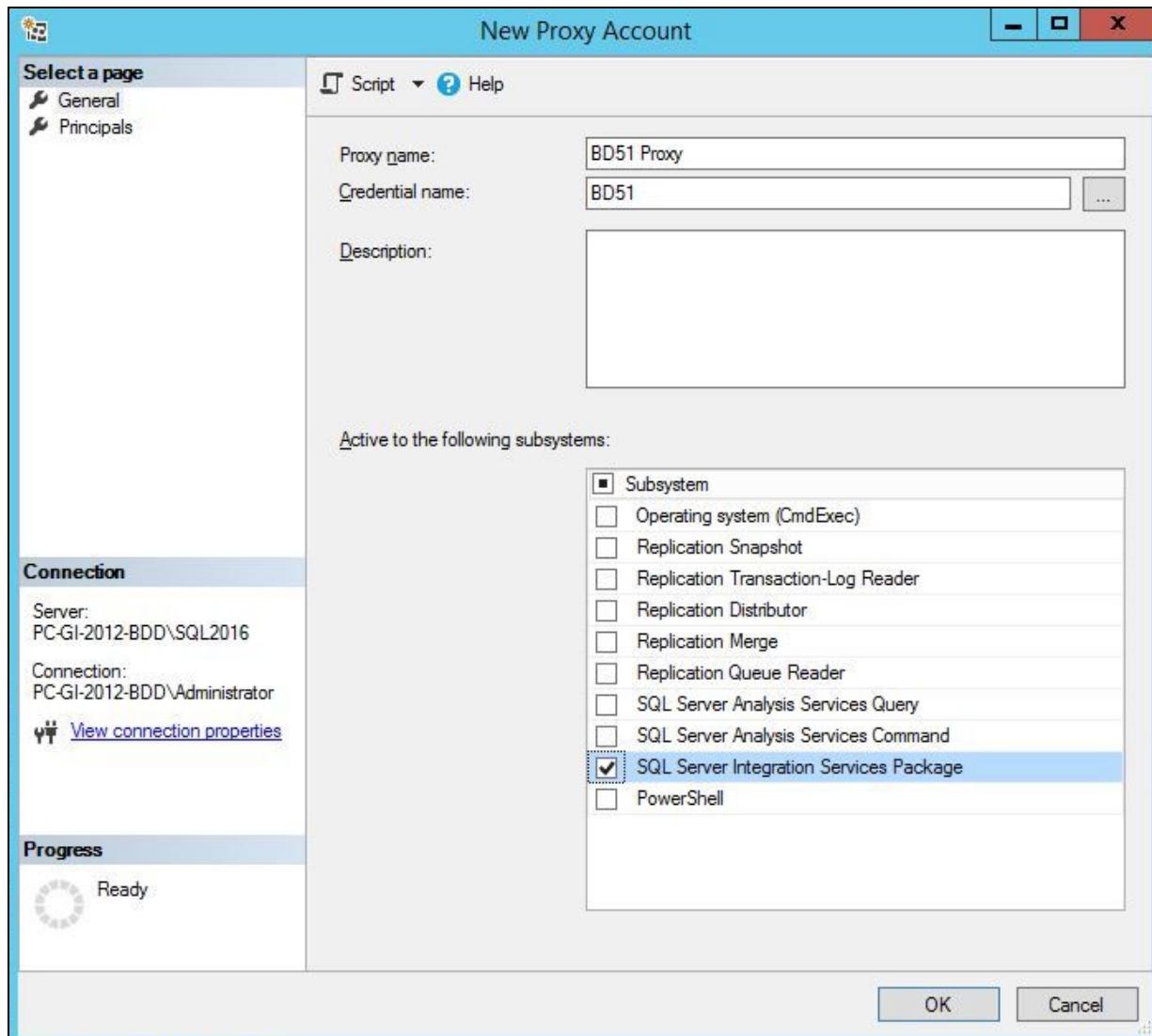


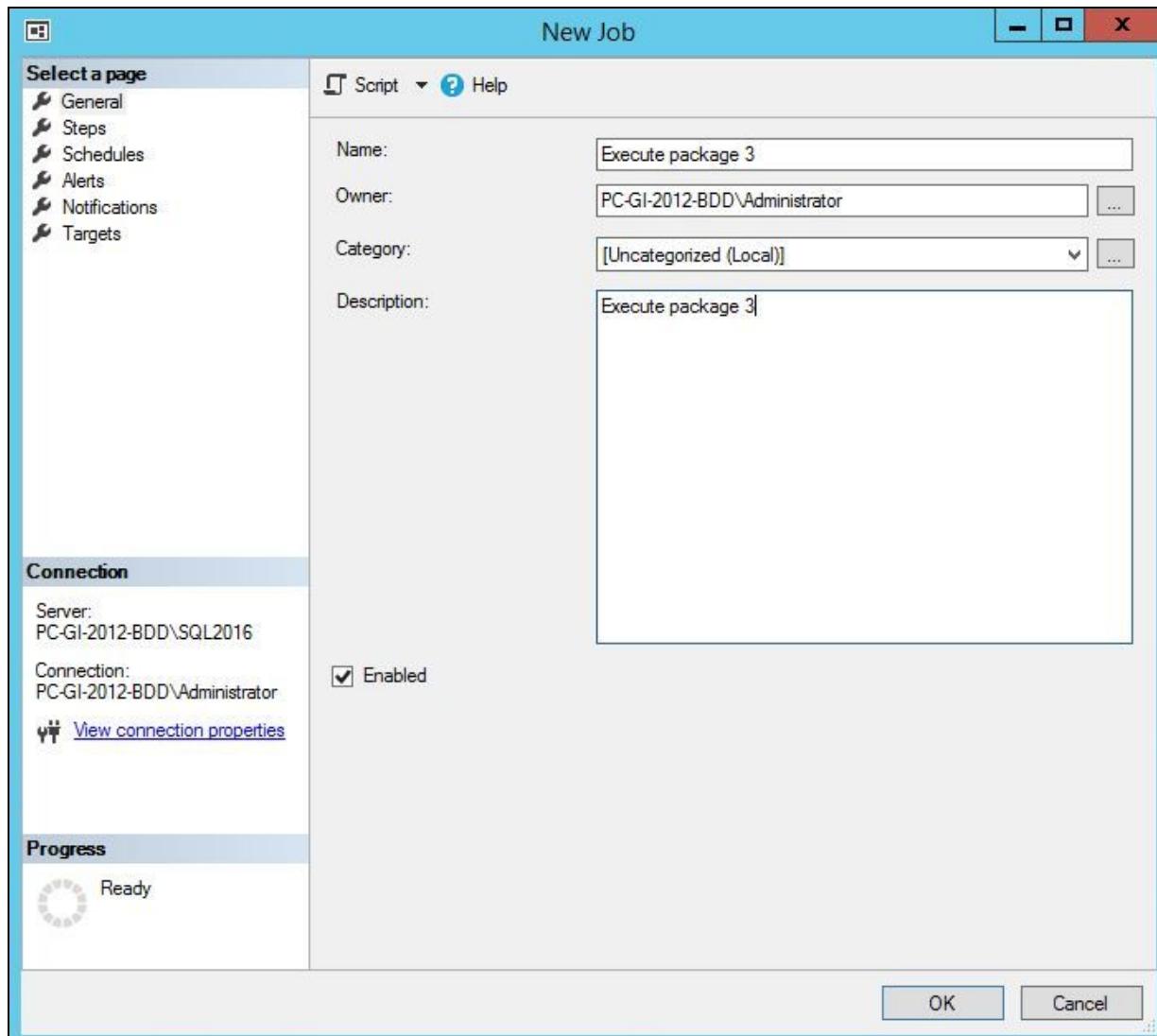
iii. Utilisation des “jobs” de SQL Server Agent

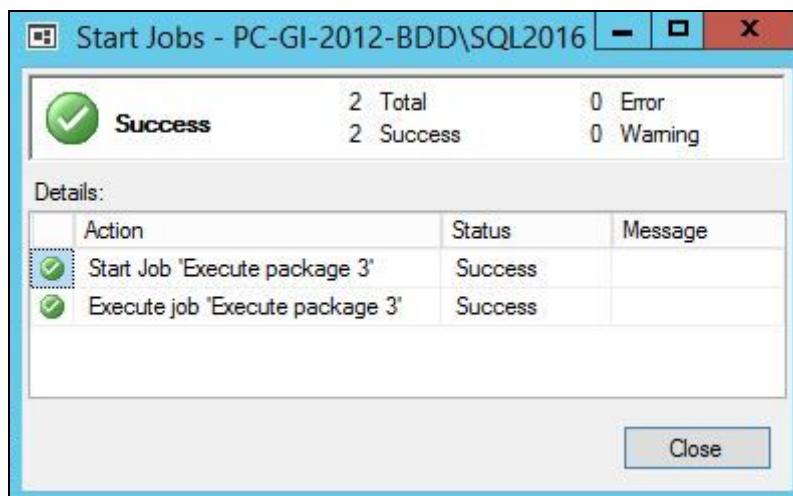
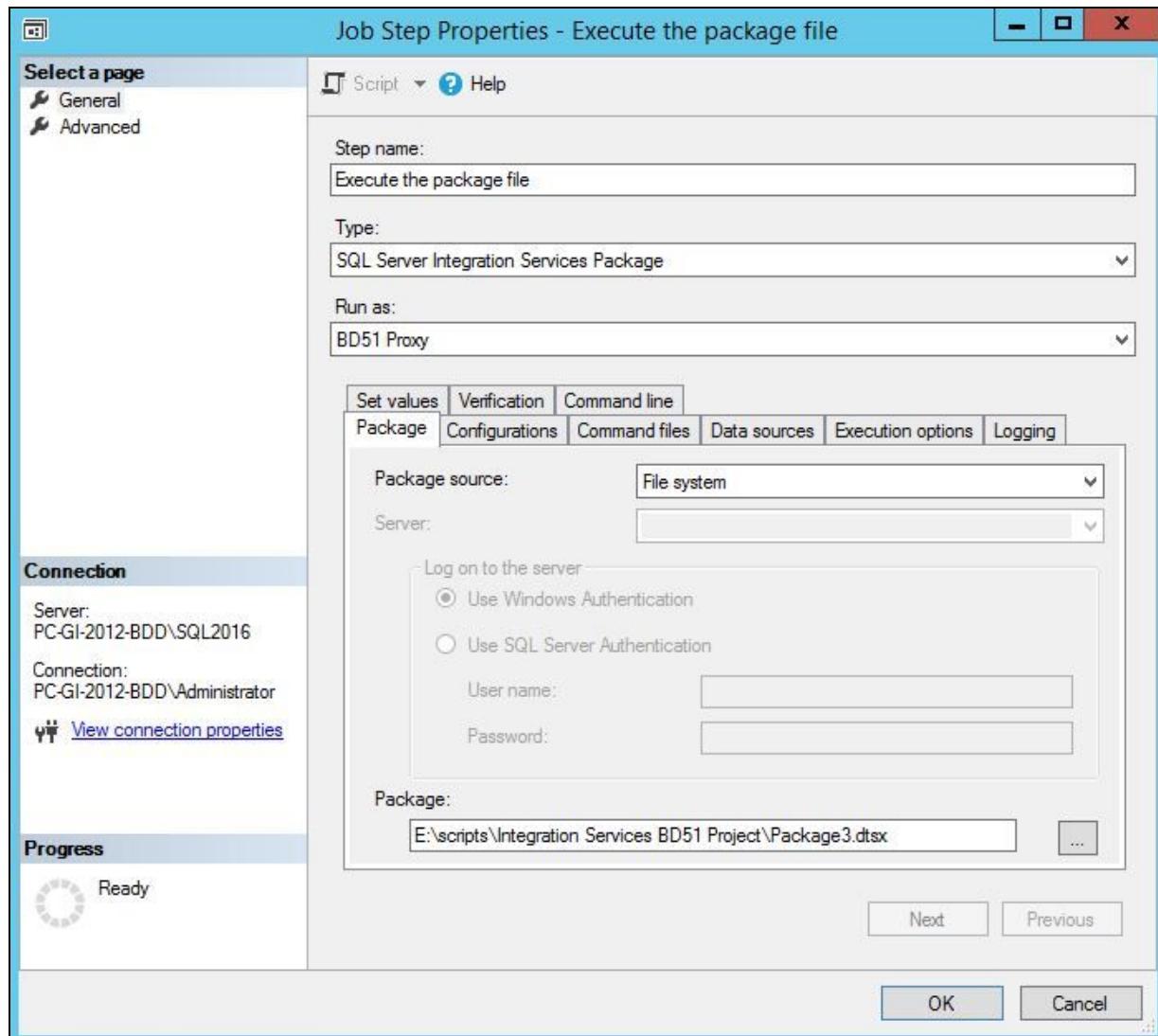
L'outil SQL Server Agent peut aussi nous servir à automatiser le processus d'ETL.

Pour ce faire nous avons commencé par créer un “Credential” que nous avons nommé “BD51” avec comme identité le compte administrateur que nous utilisons. Ensuite il a fallu créer un “Proxy” afin de permettre à notre tâche de s'exécuter avec le “Credential” précédemment créé. Enfin nous avons créé un “Job”, la tâche proprement dite.









f. Le test de l'ETL

L'exécution du test va se dérouler en deux phases :

- Dans un premier temps les tables sont vides dans la base de destination et nous effectuons le premier chargement de données via le premier package.
- Après cela des données sont insérées, modifiées, supprimées afin de tester le transfert incrémental dans les packages 2 et 3.

L'avancement de l'exécution des packages pourra être suivi grâce à l'outil de débogage dans SQL Server Data Tools.

i. Le transfert initial



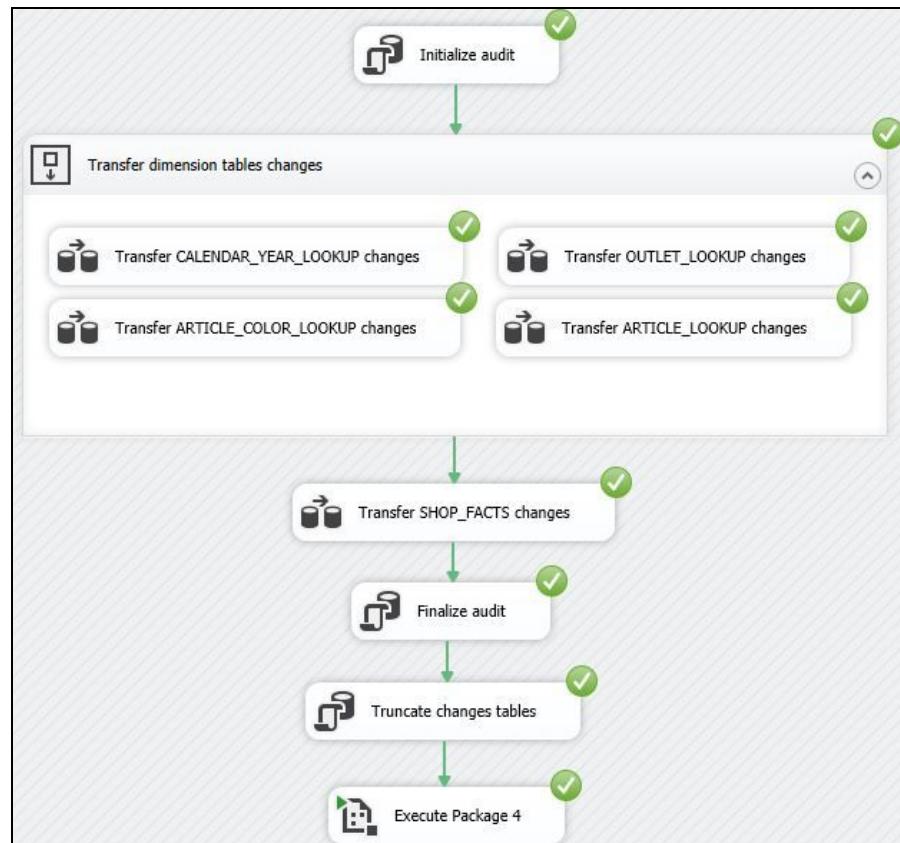
L'exécution du premier package s'est terminé sans erreur. Nous constatons que des lignes ont été envoyées dans les tables de rejets. Il s'agit de lignes possédant des clés primaires en doubles ou de lignes avec des clés étrangères inexistantes.

ii. Le transfert incrémental

Ce test va commencer par l'insertion, la modification, la suppression de données de la base de données source. Ces opérations vont déclencher les triggers qui vont créer des enregistrements dans la base de données incrémentale.

ID	ARTICLE_CODE	COLOR_CODE	WEEK_KEY	SHOP_CODE	MARGIN	AMOUNT SOLD	QUANTITY SOLD	OP_TYPE
89172	189480	901	263	353	125	151,99		1 i
89173	189481	902	263	354	100	124,99		1 i
89174	189479	119	263	353	247,3	501,5		1 i
89175	189482	902	263	353	247,3	501,5		1 i
507	115121	901	263	353	125	151,13		2 i
609	115121	189	159	3	135	298		2 i
89175	189482	902	263	353	247,3	500		1 u
10000	166136	901	175	3	86,5	500		1 u
89175	189482	902	263	353	247,3	500		1 d
10000	166136	901	175	3	86,5	500		1 d

On peut alors exécuter l'un des packages 2 ou 3 (pour rappel ces deux packages ont le même principe sauf que pour le package 2 nous avons réalisé un audit manuel et que pour le 3 nous avons utilisé la transformation *AUDIT* présente dans SSIS).



Nous pouvons constater les détails de l'exécution des opérations dans la base de destination grâce aux informations d'audit.

TRANSFER_NUM	TRANSFER_BEGIN_TIME	TRANSFER_END_TIME	INFO_NUM	TRANSFER_NUM	TABLE_NAME	ROWS_AFFECTED	OP_TYPE
1016	2019-01-07 02:49:12.417	NULL	10	1016	ARTICLE_COLOR_LOOKUP	1	u
1016	2019-01-07 02:49:12.417	NULL	11	1016	OUTLET_LOOKUP	2	i
1016	2019-01-07 02:49:12.417	NULL	12	1016	OUTLET_LOOKUP	1	u
1016	2019-01-07 02:49:12.417	NULL	13	1016	CALENDAR_YEAR_LOOKUP	4	d
1016	2019-01-07 02:49:12.417	NULL	14	1016	CALENDAR_YEAR_LOOKUP	2	i
1016	2019-01-07 02:49:12.417	NULL	15	1016	ARTICLE_LOOKUP	1	u
1016	2019-01-07 02:49:12.417	NULL	16	1016	CALENDAR_YEAR_LOOKUP	1	u

TRANSFER_NUM	TRANSFER_BEGIN_TIME	TRANSFER_END_TIME	ERROR_NUM	TRANSFER_NUM	TABLE_NAME	PK_VALUE	OP_TYPE	ERROR_CODE
1016	2019-01-07 02:49:12.417	NULL	22	1016	ARTICLE_COLOR_LOOKUP	189480901	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	23	1016	ARTICLE_COLOR_LOOKUP	189481902	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	24	1016	ARTICLE_COLOR_LOOKUP	174553901	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	25	1016	ARTICLE_COLOR_LOOKUP	158152755	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	26	1016	OUTLET_LOOKUP	354	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	27	1016	OUTLET_LOOKUP	3	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	28	1016	ARTICLE_LOOKUP	189480	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	29	1016	CALENDAR_YEAR_LOOKUP	264	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	30	1016	CALENDAR_YEAR_LOOKUP	1	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	31	1016	ARTICLE_LOOKUP	189481	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	32	1016	ARTICLE_LOOKUP	124611	i	-1071607696
1016	2019-01-07 02:49:12.417	NULL	33	1016	ARTICLE_LOOKUP	115121	i	-1071607696

Partie 2 : Optimisation de l'entrepôt de données

1. Partitionnement de la table de faits

Le partitionnement d'une table consiste à découper horizontalement cette dernière en sous-ensemble de taille plus modeste. La table est alors distribuée entre plusieurs groupes de fichiers physiques. Le procédé est totalement transparent pour l'utilisateur et le SGBD se charge de distribuer les requêtes sur les partitions concernées.

Sur les tables contenant beaucoup d'enregistrements le partitionnement permettra d'optimiser le temps de réponse à une requête, rendant ainsi plus rapide l'accès aux données. On peut aussi imaginer stocker les différentes partitions sur plusieurs disques différents, certaines données resteraient disponibles en cas de dysfonctionnement d'un disque.

Dans notre projet, une seule table pourrait nécessiter un partitionnement, la table *SHOP_FACTS* qui contient plus de 80000 lignes. Nous allons donc découper horizontalement notre table en nous basant sur le critère de temps grâce à la colonne *WEEK_KEY*. On prendra des plages de 52 pour définir une année. Le découpage se fera de la manière suivante :

Intervalle de clé	Année
1-52	1999
53-104	2000
105-156	2001
157-209	2002
210-262	2003
> 262	> 2003

a. Les groupes de fichiers

Suivant notre stratégie de partitionnement nous aurons 6 groupes de fichiers.

```
ALTER DATABASE emode ADD FILEGROUP emod_part_1999;
ALTER DATABASE emode ADD FILEGROUP emod_part_2000;
ALTER DATABASE emode ADD FILEGROUP emod_part_2001;
ALTER DATABASE emode ADD FILEGROUP emod_part_2002;
ALTER DATABASE emode ADD FILEGROUP emod_part_2003;
ALTER DATABASE emode ADD FILEGROUP emod_part_ap_2003;
```

Nous associons ensuite nos groupes de fichiers à des fichiers physiques.

```
ALTER DATABASE emode ADD FILE (NAME =emod_part_1999_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_1999_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_1999_f1

ALTER DATABASE emode ADD FILE (NAME =emod_part_2000_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_2000_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_2000

ALTER DATABASE emode ADD FILE (NAME =emod_part_2001_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_2001_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_2001

ALTER DATABASE emode ADD FILE (NAME =emod_part_2002_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_2002_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_2002

ALTER DATABASE emode ADD FILE (NAME =emod_part_2003_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_2003_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_2003

ALTER DATABASE emode ADD FILE (NAME = emod_part_ap_2003_f1,
FILENAME = 'C:\Program Files\Microsoft SQL Server 2016\MSSQL13.SQL2016\MSSQL\DATA\emod_part_ap_2003_f1.ndf'
, SIZE = 1MB, MAXSIZE = 5MB, FILEGROWTH = 2MB)
TO FILEGROUP emod_part_ap_2003
```

b. La fonction et le schéma de partition

Avec la fonction de partition nous définissons comment les lignes de notre table sont associées à un ensemble de partitions à partir des valeurs d'une colonne.

```
use emode
go
CREATE PARTITION FUNCTION Week_keyRangePFN(NUMERIC(3,0))
AS RANGE LEFT FOR VALUES (52,104,156,209,262);
```

Le schéma de partition nous permettra d'associer chaque partition définie par la fonction de partition précédente à un groupe de fichier.

```
use emode
go
CREATE PARTITION SCHEME Week_KeyRangeSCH AS PARTITION Week_keyRangePFN
TO ( emod_part_1997,emod_part_1998,emod_part_1999, emod_part_1999, emod_part_2000,emod_part_2001,emod_part_av_1990, emod_part_ap_2001)
```

c. La création de la table

Afin de prendre en compte le partitionnement, nous devons créer la table en lui spécifiant le schéma de partition créé précédemment.

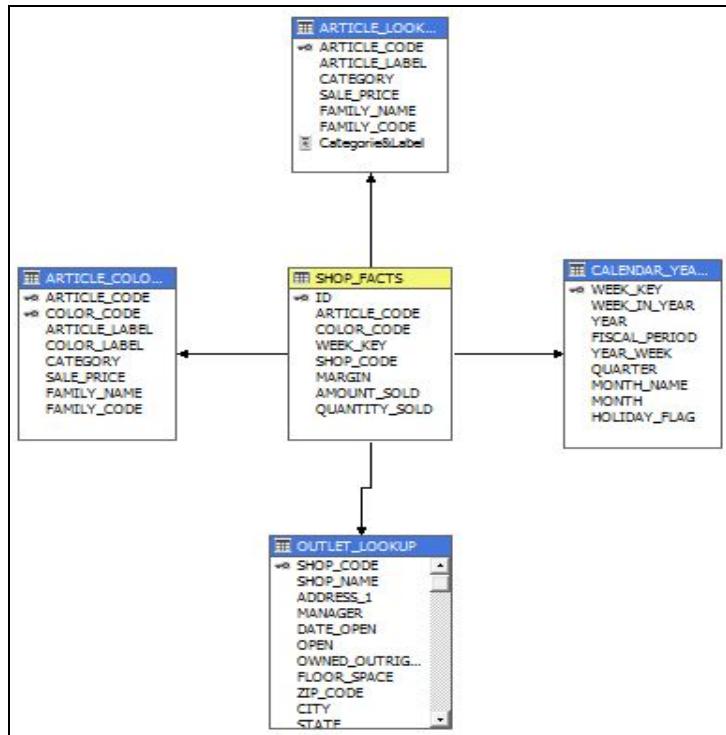
```
CREATE TABLE SHOP_FACTS
(
    ID NUMERIC (5,0) NOT NULL ,
    ARTICLE_CODE NUMERIC (6,0) ,
    COLOR_CODE NUMERIC (4,0) ,
    WEEK_KEY NUMERIC (3,0) NOT NULL,
    SHOP_CODE NUMERIC (3,0),
    MARGIN NUMERIC (18,0) ,
    AMOUNT SOLD NUMERIC (13,2) ,
    QUANTITY SOLD NUMERIC (13,2),
)
on Week_KeyRangeSCH(WEEK_KEY);
```

2. Le projet SQL Server Analysis Services (SSAS)

Ce projet consistait à exploiter un cube OLAP qui est une représentation abstraite d'informations multidimensionnelles exclusivement numérique utilisé par l'approche OLAP (acronyme de On-line Analytical Processing). Ce projet est prévu à des fins d'analyses interactives par une ou plusieurs personnes (souvent ni informaticiens, ni statisticiens) du métier que ces données sont censées représenter. Avec SQL Server Analysis Service nous avons réalisé le cube suivant avec les fonctions d'explorations de données pour les applications décisionnelles.

a. Cube OLAP

Nous avons ainsi créé le cube à partir des données de notre base de données *EMODE* sur SQL Server et obtenu le modèle en étoile suivant.



b. Dimensions

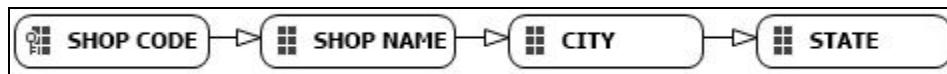
Les dimensions liées à notre cube sont les suivantes :

i. Geography

Cette dimension est liée à la table *OUTLET_LOOKUP* et se présente comme suit :

The screenshot shows the SSAS Dimension Structure tool interface. The top navigation bar includes tabs for ARTICLE COLOR LOOKUP.dim [Design], EMODE(cube [Design]), GeographyD.dim [Design] (which is selected), EMODE.ds [Design], ArticleD.dim [Design], and TemporalD.dim [Design]. The left pane, 'Attributes', contains a tree view with 'GeographyD' expanded, showing 'CITY', 'SHOP CODE', 'SHOP NAME', and 'STATE'. The middle pane, 'Hierarchies', shows a 'Hierarchy' tree with 'STATE' at the top, followed by 'CITY', 'SHOP NAME', and 'SHOP CODE'. A tooltip indicates 'To create a new hierarchy, drag an attribute here.' The right pane, 'Data Source View', displays the columns of the 'OUTLET_LOOKUP' table, including SHOP_CODE, SHOP_NAME, ADDRESS_1, MANAGER, DATE_OPEN, OPEN, OWNED_OUTRIGHT, FLOOR_SPACE, ZIP_CODE, CITY, and STATE.

La relation qui relie les différents attributs de cette dimension est la suivante:



Après le déploiement du projet nous avons accès aux données de la table *OUTLET_LOOKUP* :

The screenshot shows the SSAS Solution Explorer interface. The top navigation bar includes tabs for ARTICLE COLOR LOOKUP.dim [Design], EMODE(cube [Design]), GeographyD.dim [Design] (selected), EMODE.ds [Design], ArticleD.dim [Design], and TemporalD.dim [Design]. The left pane shows a tree view of the deployed solution, including 'Allemande' (Germany) with 'Berlin' and 'e-Mode Berlin' nodes, and other countries like Belgique, Espagne, France, Grande-Bretagne, Italie, Monaco, Pays-Bas, and Unknown. The right pane, 'Explorateur de solutions', displays the 'Solution BD51ProjectN (1 projet)' node, which contains 'BDS1ProjectN' (Data Sources: EMODE.ds), 'Data Source Views: EMODE.ds', 'Cubes: EMODE(cube)', 'Dimensions: TemporalD.dim, ArticleD.dim, GeographyD.dim (selected)', 'Mining Structures', 'Roles', 'Assemblies', and 'Miscellaneous'.

ii. Temporal

Cette dimension est liée à la table *CALENDAR_YEAR_LOOKUP*. En se basant sur la présentation faite précédemment l'architecture, les relations entre les attributs de la dimensions et les données présentées suite au déploiement sont les suivantes:



iii. Article

Cette dimension se base sur la table *ARTICLE_LOOKUP*.



Partie 3 : Implémentation du reporting

1. Le projet SQL Server Reporting Services (SSRS)

Pour le reporting avec SSRS nous avons récupéré les données et réalisé les différents reports à partir du cube. Nous avons donc réalisés plusieurs reports par rapport aux montant des ventes en un premier temps un report pour le montant **total de ventes par magasin et par années** puis dans un second temps nous avons réalisé un report **par années par magasins, par catégories et par articles** et pour finir nous avons réalisé un report **comparatif du pourcentage d'évolution du total des ventes par magasin par années**.

a. Total des ventes par magasins par années

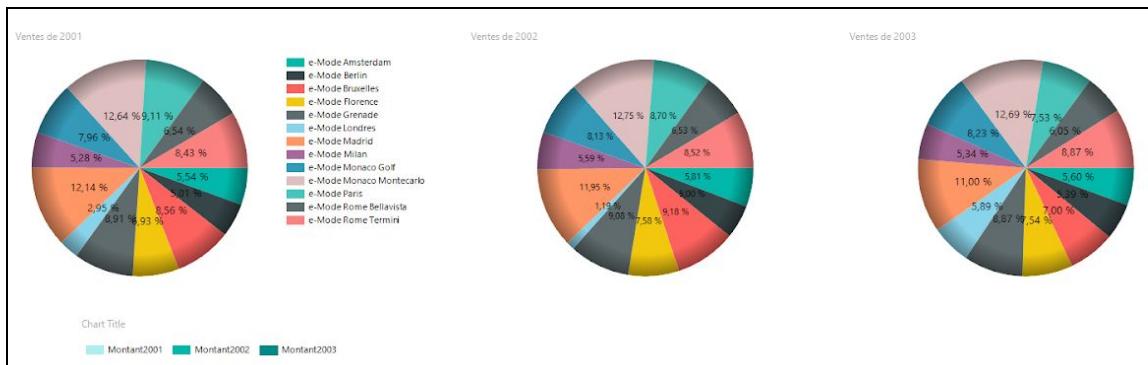
Magasin	2001	2002	2003
e-Mode Amsterdam	448301,50	768389,50	843584,20
e-Mode Berlin	405985,10	661249,80	811923,60
e-Mode Bruxelles	693210,50	1215158,00	1053581,40
e-Mode Florence	561123,40	1003070,70	1135479,10
e-Mode Grenade	721573,70	1201063,50	1336003,30
e-Mode Londres	238818,70	157718,70	887169,20
e-Mode Madrid	982637,10	1581616,00	1656675,70
e-Mode Milan	427244,70	739368,70	803420,80
e-Mode Monaco Golf	644635,10	1076144,00	1239587,40
e-Mode Monaco Montecarlo	1023060,70	1687359,10	1911434,30
e-Mode Paris	737914,20	1150658,80	1134085,40
e-Mode Rome Bellavista	529078,50	863653,10	910451,20
e-Mode Rome Termini	682230,80	1126796,10	1335747,20
Total	8095814,00	13232246,00	15059142,80

b. Total des ventes par magasins, par catégories et par articles

SHOP NAME	CATEGORY	ARTICLE LABEL	Quantity	Montant	
e-Mode Grenade	Vêtements d'intérieur	Caleçon à rayures	34,00	8220,80	
		Caleçon en lycra	35,00	4537,40	
		Chemise style Rudolph	34,00	5323,60	
		Fume-cigarette à diamants	15,00	2181,40	
		Maillot académique en lycra	19,00	2587,30	
		T-Shirt Jacquard	41,00	6811,50	
		T-Shirt moulant en chenille	38,00	6502,30	
		Tunique ceinturée	1,00	79,00	
		Veste en soie à double boutonnage	45,00	6400,20	
		Veste zippée	21,00	1254,80	
			294,00	44352,50	
Total			19830,00	3258640,50	
e-Mode Londres	2 chemises avec poche	Chemise en popeline avec poche	18,00	3905,20	
		Chemise milanaise en maille	233,00	44801,50	
		Chemise style militaire	38,00	7418,90	
			289,00	58225,80	
	Accessoires pour les chev.	Peigne rond	128,00	25873,80	
			128,00	25873,80	
	Bermuda	Short en velours côtelé	25,00	3927,70	
			25,00	3927,70	
	Bijouterie	Collier de perles à deux rangs	10,00	1511,40	
			10,00	1511,40	
	Cardigan	Cardigan en chevron	4,00	532,70	
			4,00	532,70	

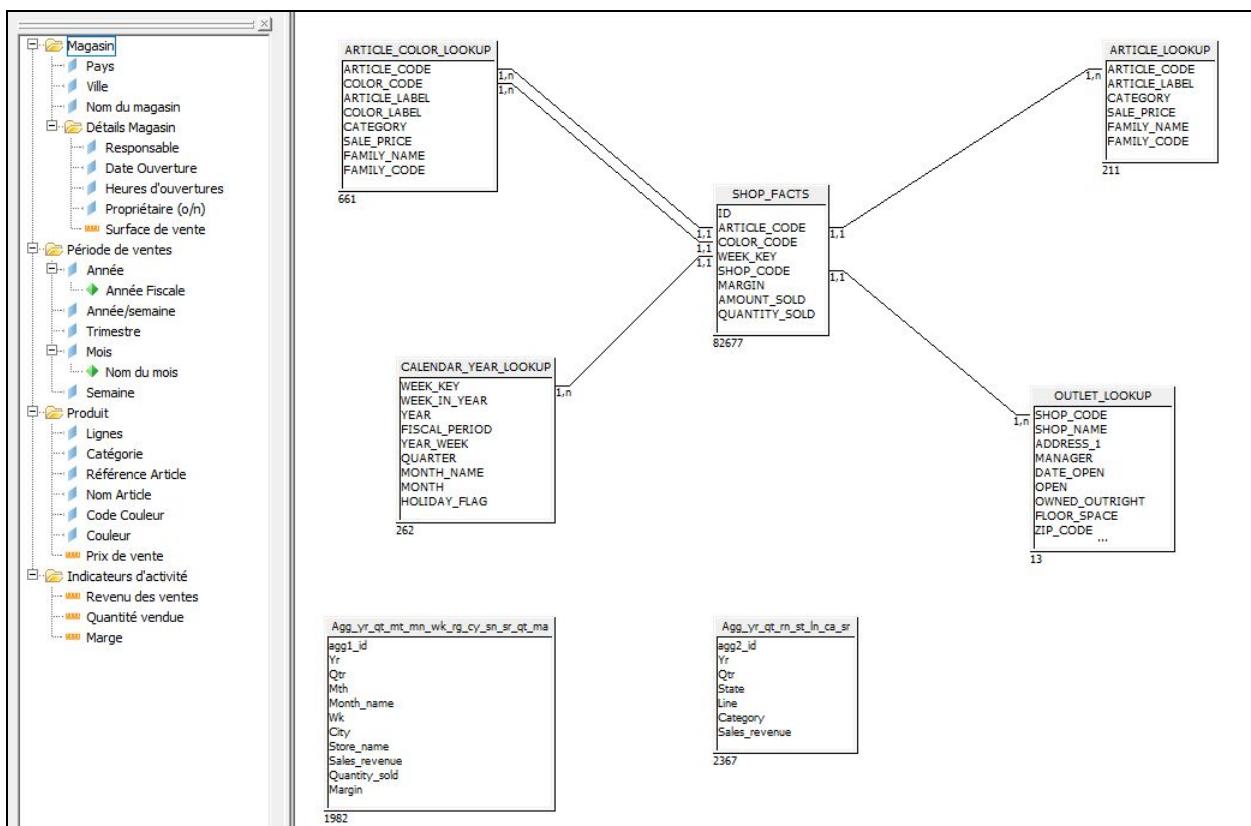
c. Comparatif de dépenses par magasins entre les années

Vente par année par magasins			
Shop name	Montant 2001	Montant 2002	Montant 2003
e-Mode Bruxelles	693210,50	1215158,00	1053581,40
e-Mode Rome Termini	682230,80	1126796,10	1335747,20
e-Mode Milan	427244,70	739368,70	803420,80
e-Mode Madrid	982637,10	1581616,00	1656675,70
e-Mode Londres	238818,70	157718,70	887169,20
e-Mode Rome Bellavista	529078,50	863653,10	910451,20
e-Mode Monaco Montecarlo	1023060,70	1687359,10	1911434,30
e-Mode Amsterdam	448301,50	768389,50	843584,20
e-Mode Paris	737914,20	1150658,80	1134085,40
e-Mode Monaco Golf	644635,10	1076144,00	1239587,40
e-Mode Florence	561123,40	1003070,70	1135479,10
e-Mode Grenade	721573,70	1201063,50	1336003,30
e-Mode Berlin	405985,10	661249,80	811923,60



2. Création de l'univers Business Objects (BO)

Nous avons créé notre BO Universe avec **Designer** sur la machine locale. Le serveur utilisé est **PC-GI-405\SQL2014** avec la base de donnée **Emode** sur le compte **bd51/bd51**. Ci-après notre BO Universe :



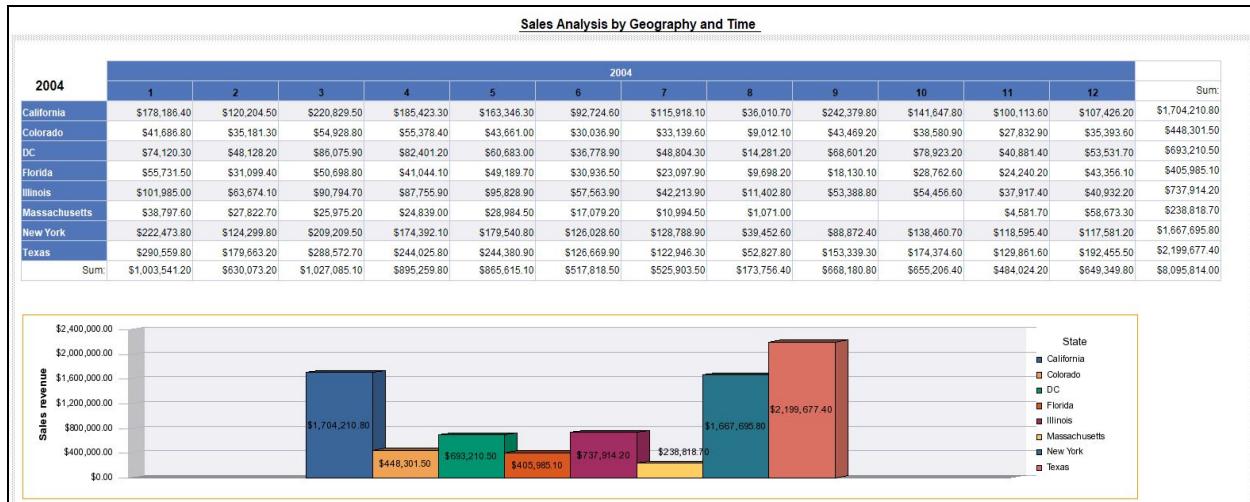
3. Reporting avec Web Intelligence

La réalisation des rapports avec Web Intelligence a nécessité un Univers existant. Cependant nous avions eu le choix entre celui créé précédemment et eFashion Universe. Nous avions donc choisi **eFashion Universe** pour produire les rapports suivants :

a. Analyse des ventes par temps et situation géographique

Sales Analysis by Geography and Time													
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	\$42,469.20	\$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	\$68,801.20	\$78,923.20	\$40,881.40	\$53,531.70	
Florida	\$55,731.50	\$31,099.40	\$50,998.80	\$41,044.10	\$49,189.70	\$30,936.50	\$23,097.90	\$9,598.20	\$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	
2005	2005												
	1	2	3	4	5	6	7	8	9	10	11	12	
California	\$228,267.50	\$128,909.10	\$295,538.20	\$220,477.70	\$182,835.10	\$125,943.10	\$149,158.50	\$147,114.80	\$464,168.30	\$327,991.20	\$202,222.50	\$312,053.50	
Colorado	\$70,036.30	\$41,300.70	\$77,794.40	\$64,917.70	\$54,591.90	\$37,827.50	\$47,109.10	\$28,042.20	\$117,115.80	\$90,613.40	\$56,225.20	\$82,185.30	
DC	\$95,960.30	\$55,960.80	\$127,568.40	\$99,078.30	\$93,787.60	\$70,619.90	\$61,389.00	\$57,886.00	\$169,651.20	\$159,384.80	\$95,600.90	\$128,270.80	
Florida	\$73,919.00	\$31,270.60	\$69,086.20	\$53,238.20	\$59,210.80	\$34,909.30	\$39,069.50	\$29,444.40	\$52,600.50	\$77,762.40	\$58,074.40	\$82,464.50	
Illinois	\$156,927.30	\$57,663.10	\$119,706.40	\$100,256.90	\$96,982.70	\$57,482.40	\$66,726.20	\$48,572.10	\$115,275.10	\$142,000.10	\$77,616.00	\$111,450.50	
Massachusetts												\$32,289.10	\$125,428.60
New York	\$292,888.80	\$113,226.20	\$277,856.00	\$227,484.90	\$283,938.40	\$181,089.30	\$210,746.50	\$78,362.60	\$212,110.90	\$314,297.90	\$259,748.20	\$311,753.40	
Texas	\$419,402.70	\$180,682.30	\$414,207.90	\$302,855.20	\$310,538.30	\$182,585.60	\$227,755.90	\$191,471.40	\$365,333.00	\$433,822.00	\$300,139.00	\$404,095.30	
2006	2006												
	1	2	3	4	5	6	7	8	9	10	11	12	
California	\$288,259.70	\$161,506.40	\$279,979.00	\$260,419.70	\$327,340.50	\$201,637.50	\$250,330.40	\$156,525.00	\$368,910.90	\$298,916.00	\$202,128.30	\$196,725.60	
Colorado	\$82,188.40	\$49,653.30	\$72,912.20	\$71,550.30	\$81,342.00	\$60,770.30	\$64,337.50	\$49,775.40	\$118,776.00	\$80,313.40	\$59,011.30	\$52,954.10	
DC	\$108,675.00	\$73,419.80	\$96,913.00	\$94,271.50	\$98,438.10	\$70,388.10	\$91,387.30	\$47,496.40	\$132,761.60	\$102,744.30	\$84,697.10	\$52,389.20	
Florida	\$85,676.50	\$42,191.90	\$76,013.80	\$64,080.20	\$88,730.80	\$70,658.20	\$67,346.40	\$45,599.70	\$102,623.00	\$73,108.60	\$49,371.80	\$48,522.70	
Illinois	\$116,259.60	\$53,409.70	\$85,988.50	\$90,182.50	\$170,065.80	\$94,476.00	\$71,732.80	\$54,914.60	\$146,538.90	\$81,115.20	\$91,692.80	\$77,709.00	

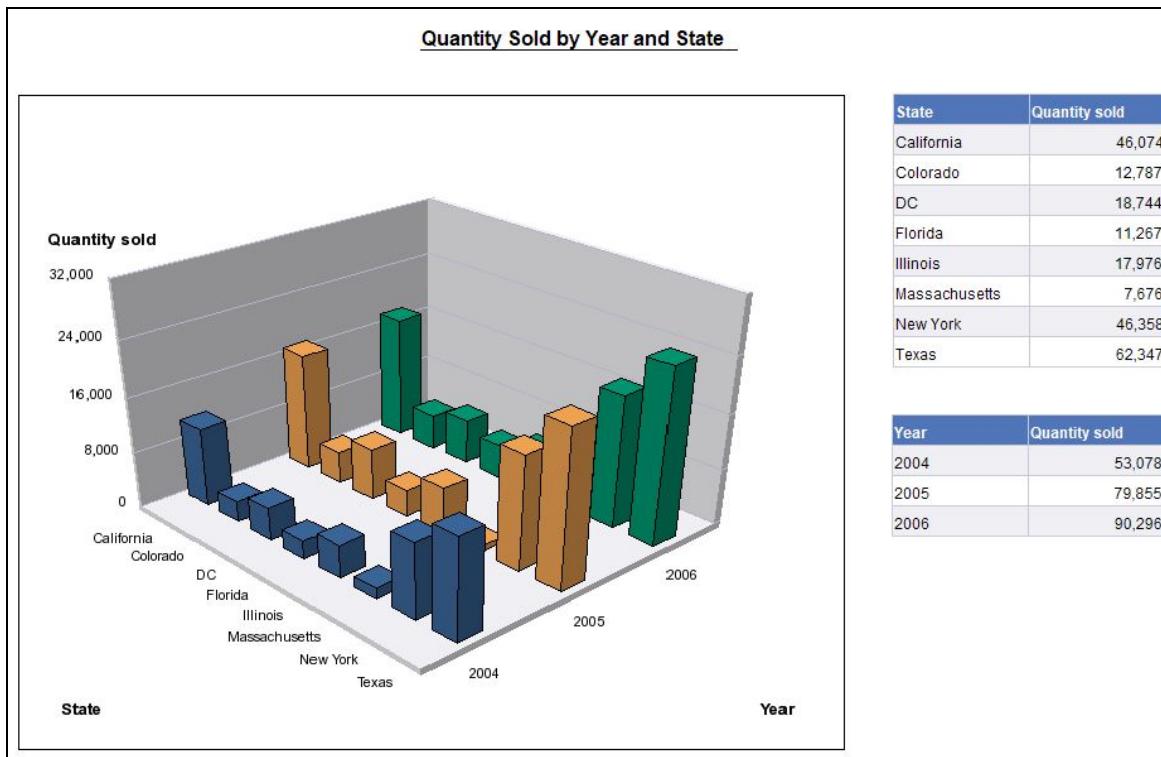
b. Analyse des ventes par temps et situation géographique avec les totaux



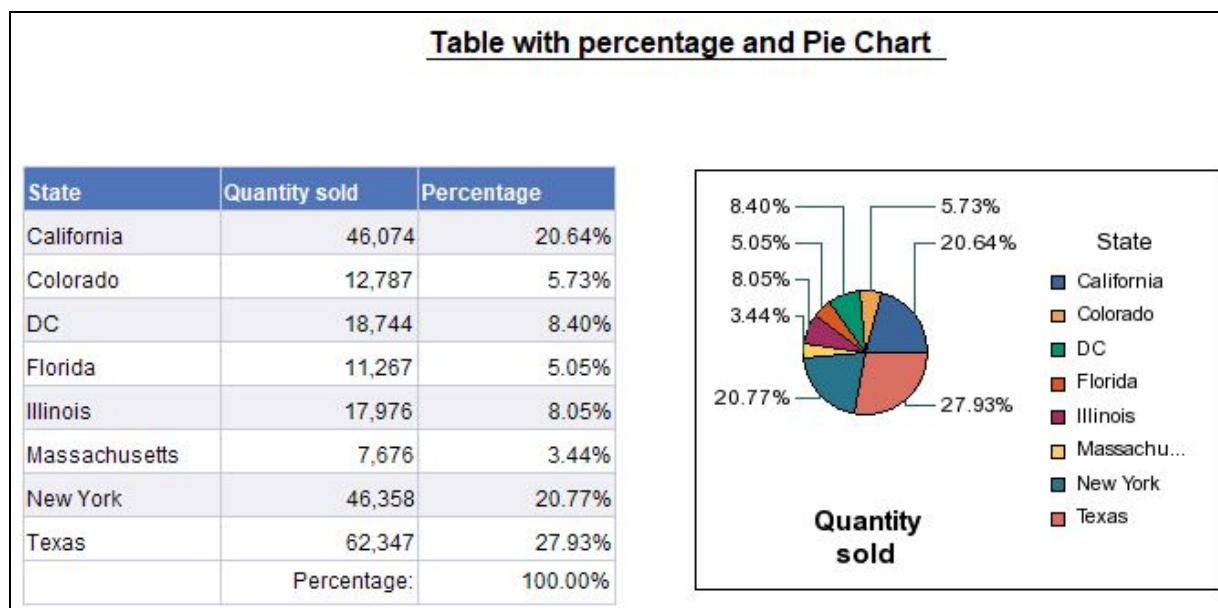
c. Quantité vendue par année et par état

Quantity Sold by Year and State		
Year	State	Quantity sold
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
2004	S/Total :	53,078
Year	State	Quantity sold
2005	California	17,001
	Colorado	4,700
	DC	7,572
	Florida	3,852
	Illinois	6,744
	Massachusetts	902
	New York	16,447
	Texas	22,637
2005	S/Total :	79,855

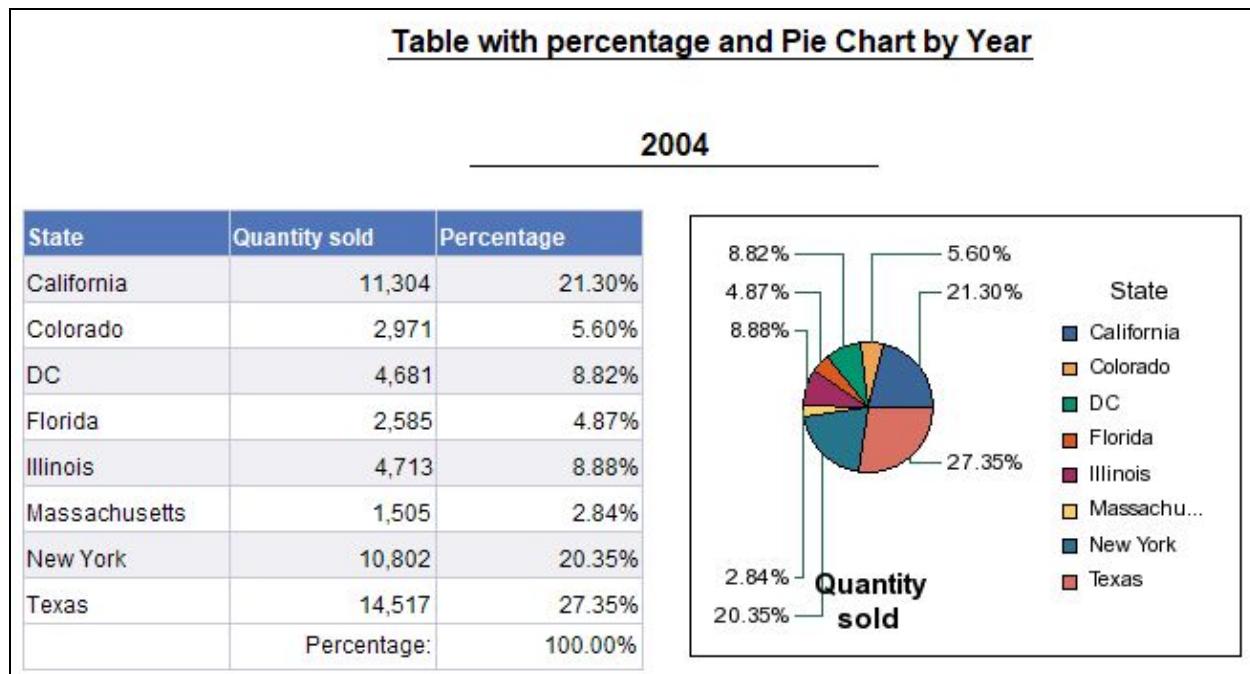
d. Graphique des quantités vendues en 3D histogram



e. Graphique des quantités vendues et pourcentage en Pie Chart



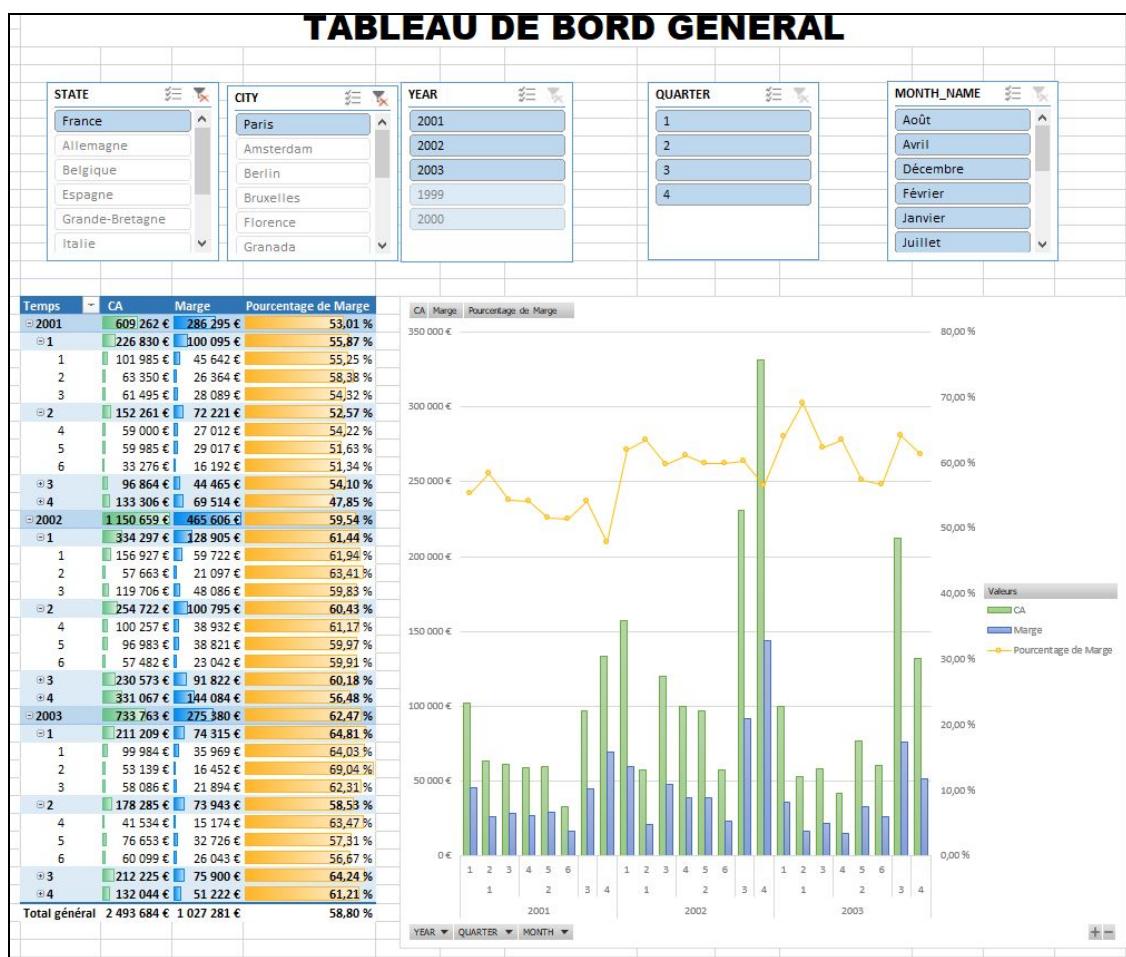
f. Graphique des quantités vendues et pourcentage par année Pie Chart



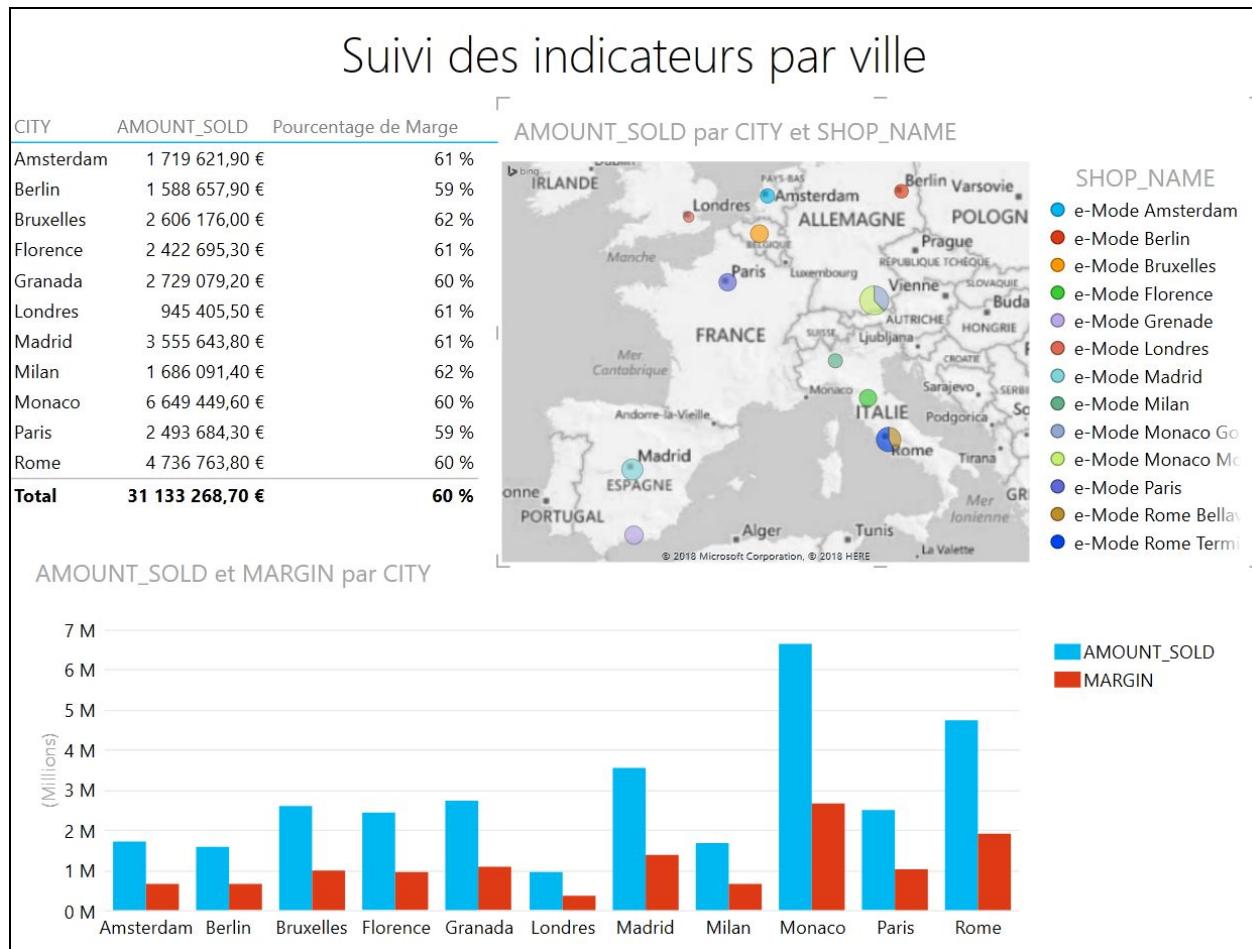
4. Reporting avec Microsoft Excel

Pour le reporting avec Excel nous avons récupéré les données à partir du cube et nous avons réalisé les différents reports avec Power Pivot. Nous avons donc mis en place en un premier temps un tableau de bord pour différentes analyses puis en un second temps un autre tableau de suivi des indicateurs géographiquement par ville.

a. Tableau de bord général



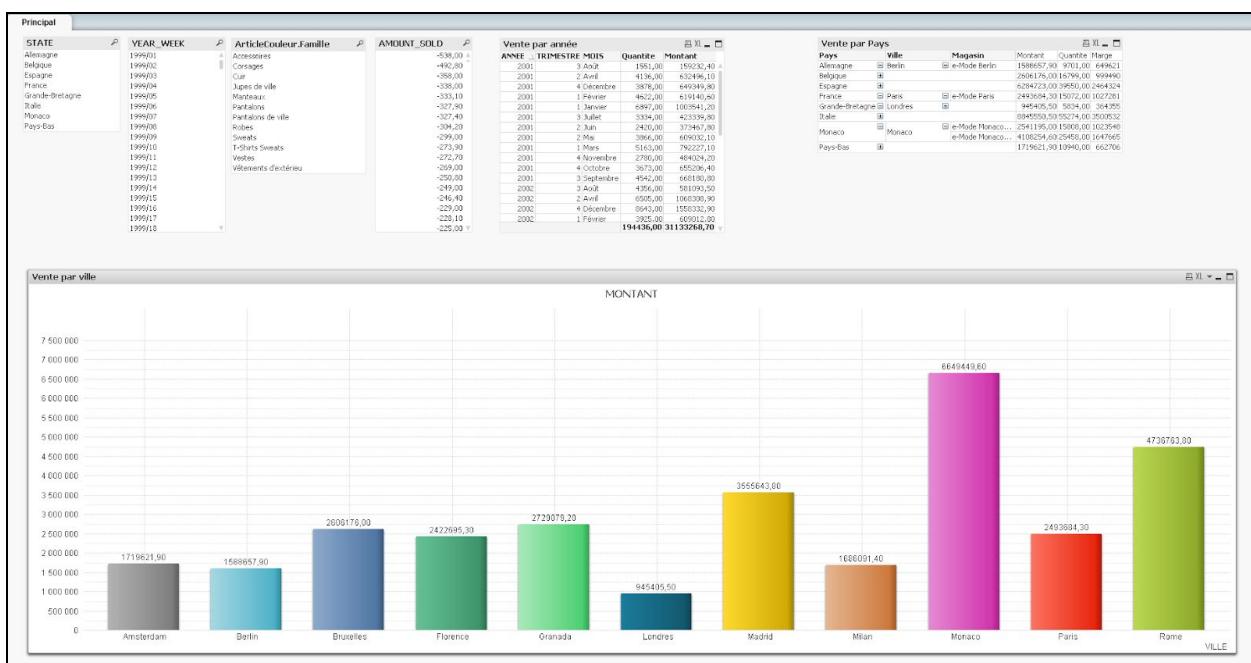
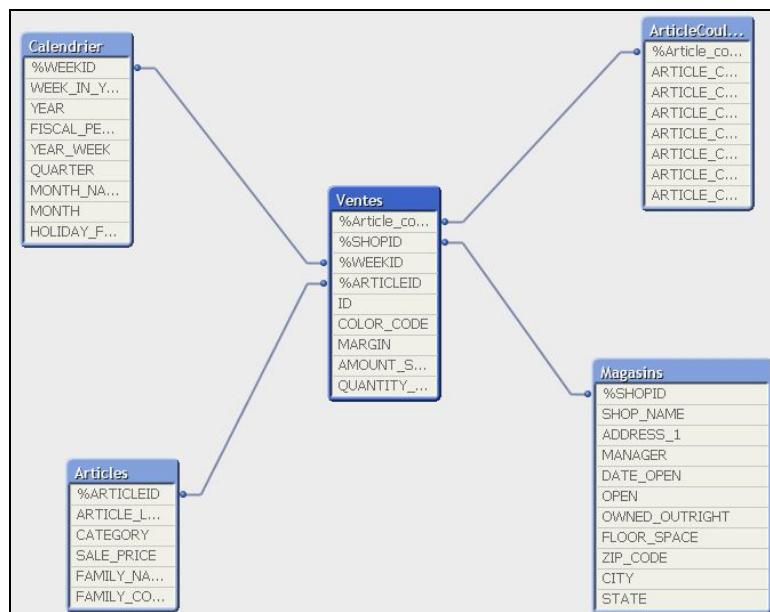
b. Suivi des indicateurs par ville



5. Reporting avec Qlikview

Qlikview est un outil BI très pratique. Il permet de faire à la fois de l'ETL et du Reporting.

Nous l'avons alors utilisé pour nous connecter à notre base Emode sur SQL Server, charger les données, créer notre modèle en étoile et faire un tableau de reporting. Le modèle en étoile et le tableau de bord de reporting sont présentés dans les figures ci-dessous.



Conclusion

Au terme de ce projet nous pouvons dire que sa réalisation a été pour nous l'occasion de mettre en pratique l'ensemble des notions vues en cours de l'UV **BD51**, aussi bien en cours qu'en travaux pratiques.

Ce projet nous a aussi permis de nous préparer à aborder les différents projets technologiques auxquels nous seront amenés à participer dans le futur en matière de systèmes décisionnels.

Par ailleurs, grâce à ce projet nous avons pu appréhender des concepts qui nous étaient jusque là étrangers et maîtriser bon nombre d'outils innovants indispensables dans les secteurs de pointes du monde de l'informatique notamment le BI.

En plus des compétences techniques et pratiques acquises, ce projet a accru nos connaissances en matière d'organisation du travail et renforcé nos capacités de collaboration à travailler en équipe.