

**Création du document : Stéphanie GORGONE – Chloé MANDON**

**Groupe Restitution Jasper**

**Date : 12/02/2011**

Rapport technique jasper

Projet Darties – Groupe2

Rapport technique jasper

Projet Darties – Groupe2

# Contenu

[I. Contenu 1](#_Toc285830102)

[II. Introduction 2](#_Toc285830103)

[III. Création d’un document jasper 3](#_Toc285830104)

[1. Les tableaux 3](#_Toc285830105)

[Création de modeles de tableaux 3](#_Toc285830106)

[Les différentes zones d’un rapport 5](#_Toc285830107)

[Respect du SFD 5](#_Toc285830108)

[2. Les graphiques 6](#_Toc285830109)

[Creation de modeles de graphiques 6](#_Toc285830110)

[Respect du SFD 6](#_Toc285830111)

[IV. Intégration des tableaux et graphiques en Java 7](#_Toc285830112)

[1. Choix du fichier Jasper 7](#_Toc285830113)

[2. Chargement et compilation d’un rapport jrxml 8](#_Toc285830114)

[3. Changement de requêtes 8](#_Toc285830115)

[Changement de la clause *WHERE* de la requête 8](#_Toc285830116)

[CHANGement de la clause *select* de la requete 9](#_Toc285830117)

[4. Passage des paramètres 10](#_Toc285830118)

[Titres et sous-titres des rapports 10](#_Toc285830119)

[Periodes dans l’entete des tableaux 10](#_Toc285830120)

[V. Conclusion 11](#_Toc285830121)

# Introduction

iReport est un éditeur graphique permettant de concevoir des modèles de rapport. Ce logiciel Open Source fonctionne à partir de la librairie JasperReports. Il permet de créer différents types de rapports (tableaux, graphiques, etc.) et d’en visualiser le rendu.

La création d’un rapport permet d’obtenir un fichier XML (fichier jrxml). Nous utilisons ces fichiers XML dans notre application pour afficher les tableaux et graphiques. En effet, un dossier *« TableauxXML »* comporte tous les fichiers XML des tableaux et graphiques créés sous iReport. Ensuite, nous utilisons les objets JAVA qui permettent la génération d’un rapport (ces derniers seront détaillés dans la deuxième partie du rapport).

Les données des tableaux ou des graphiques seront alimentées par une base de données. La sélection de ces données se fera par une requête associée à chacun des tableaux ou graphiques.

Pour simplifier la création de ces différents tableaux, nous avons choisi de créer un tableau pour chaque couple indicateur-caractéristiques. Plus précisément, pour chaque onglet, nous avons un tableau pour un indicateur et une caractéristique (par exemple un tableau CA-toutes caractéristiques, un autre CA-Réalisé etc.). Si un onglet permet de voir toutes les caractéristiques (Objectifs, Réalisé, Toutes) et tous les indicateurs (CA, Ventes, Marges, Toutes), il y aura 12 modèles de rapport créés (en fichier jrxml).

# Création d’un document jasper

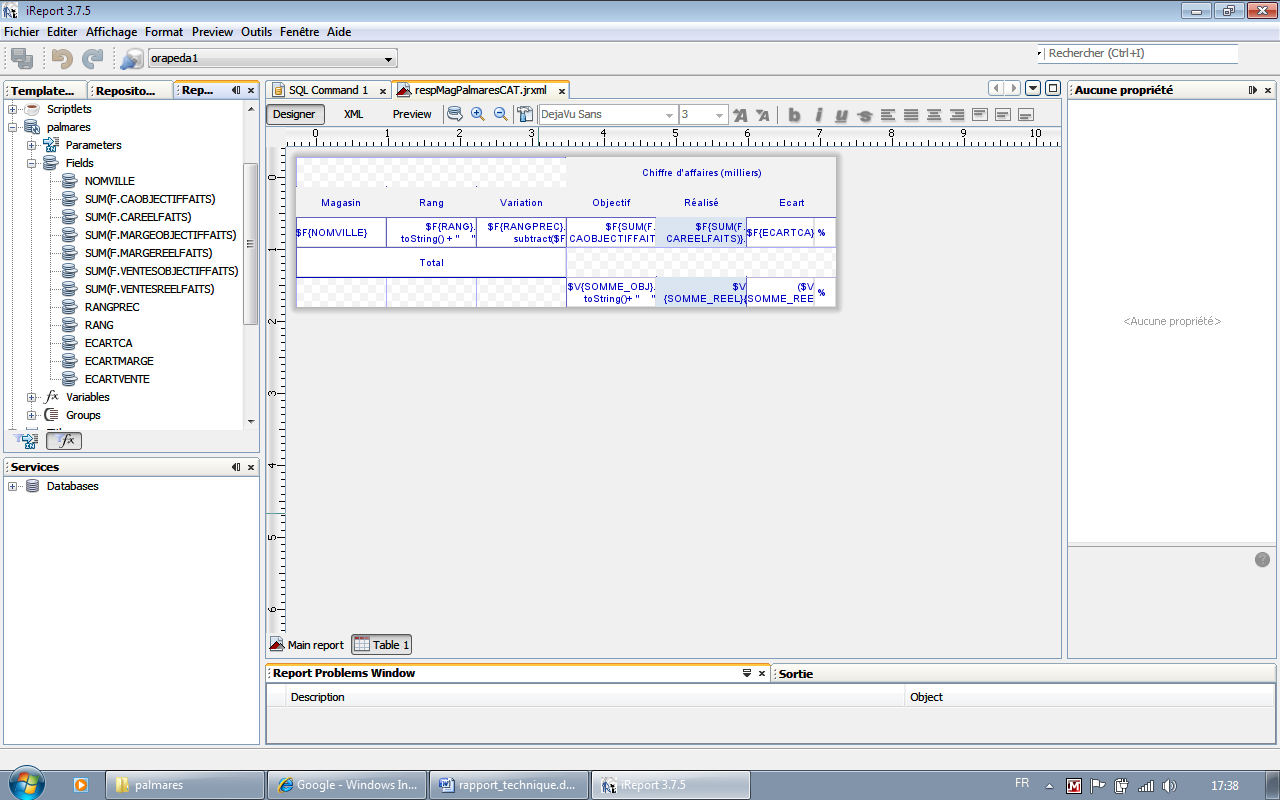
## Les tableaux

### Création de modeles de tableaux

La première étape de création d’un rapport consiste à créer un document vide avec une requête SQL générale (sans quoi la génération du rapport ne fonctionne pas).

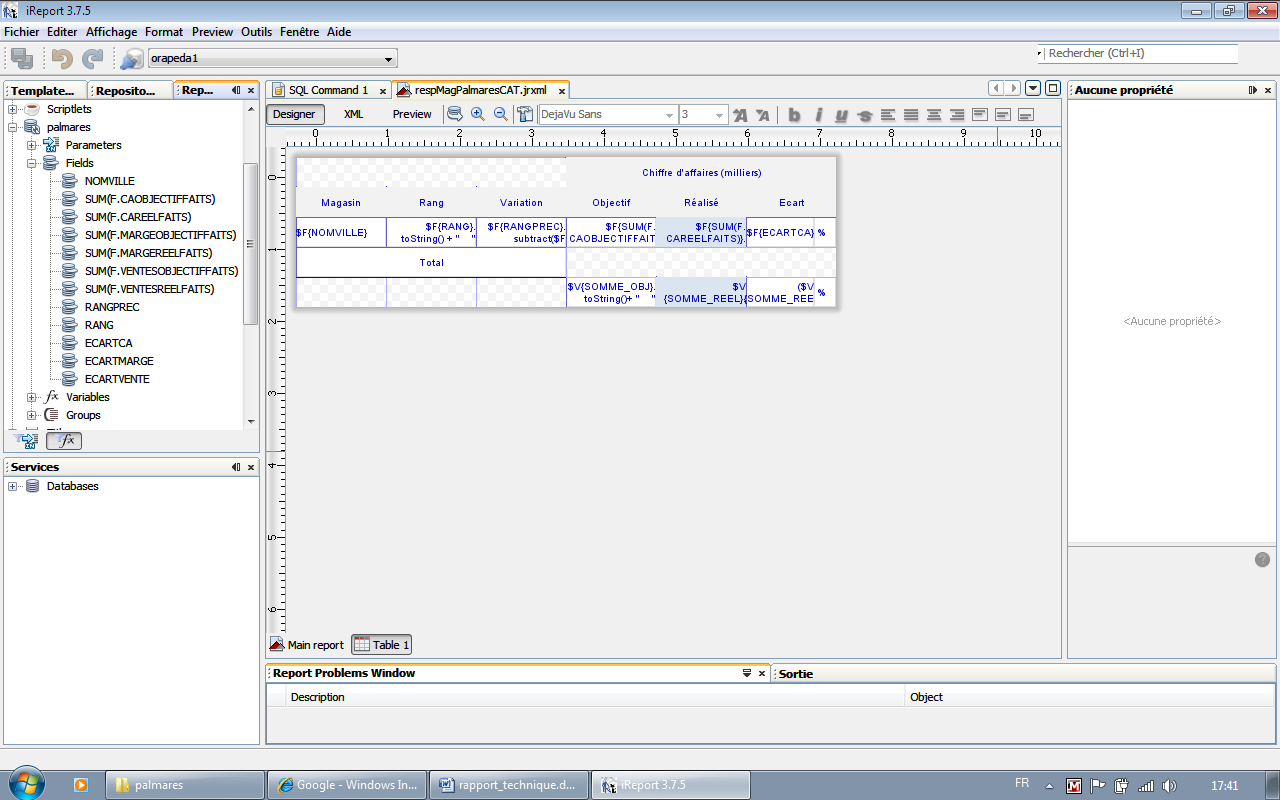
La liaison entre un tableau et la base de données se fait via un dataset (ensemble de données récupérées par partir d’une requête SQL).

La création d’un tableau se fait à l’aide du dataset. Il permet de récupérer les champs de la base de données que l’on veut afficher dans le tableau.



Le dataset *palmares* a été créé et permet de récupérer les champs résultat *(Fields)* de la requête SQL.

Ensuite on peut remplir le tableau en associant une expression (champs résultant de la requête ou champs saisissables) à chaque case de celui-ci.

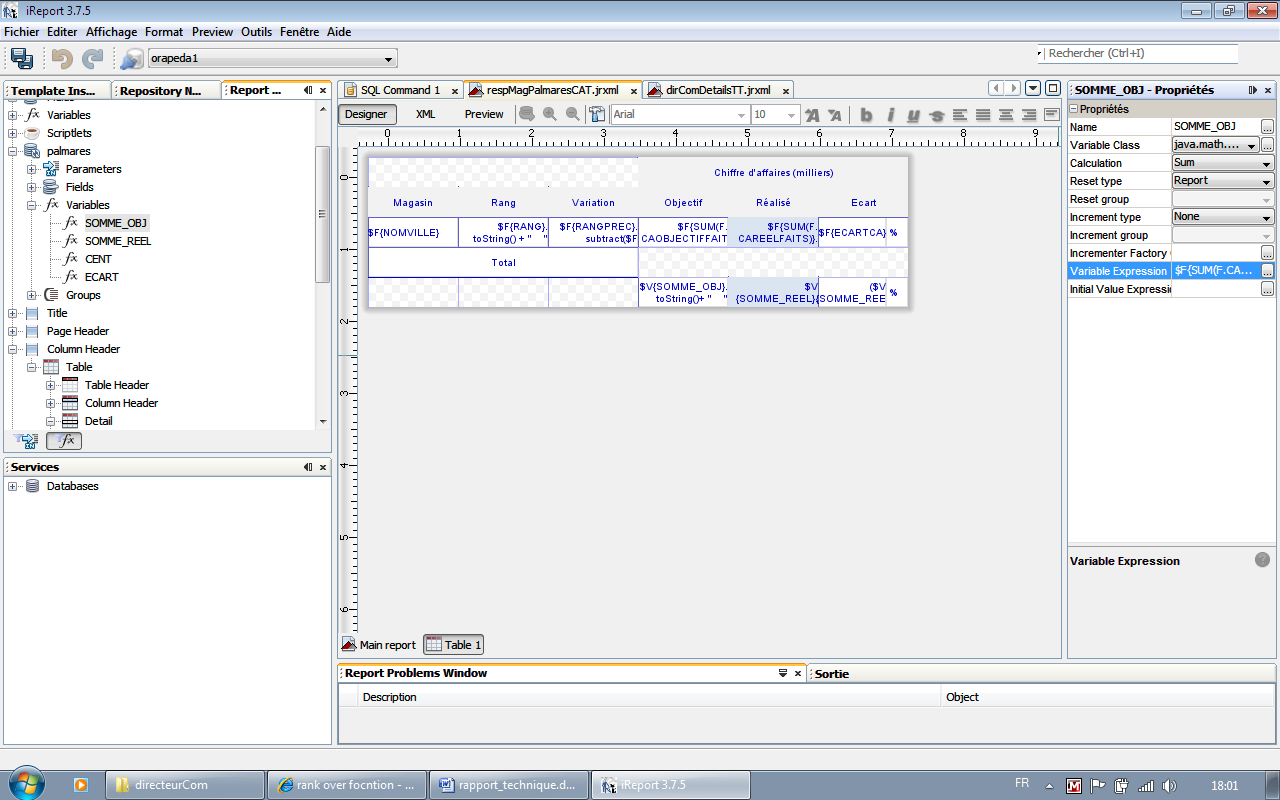
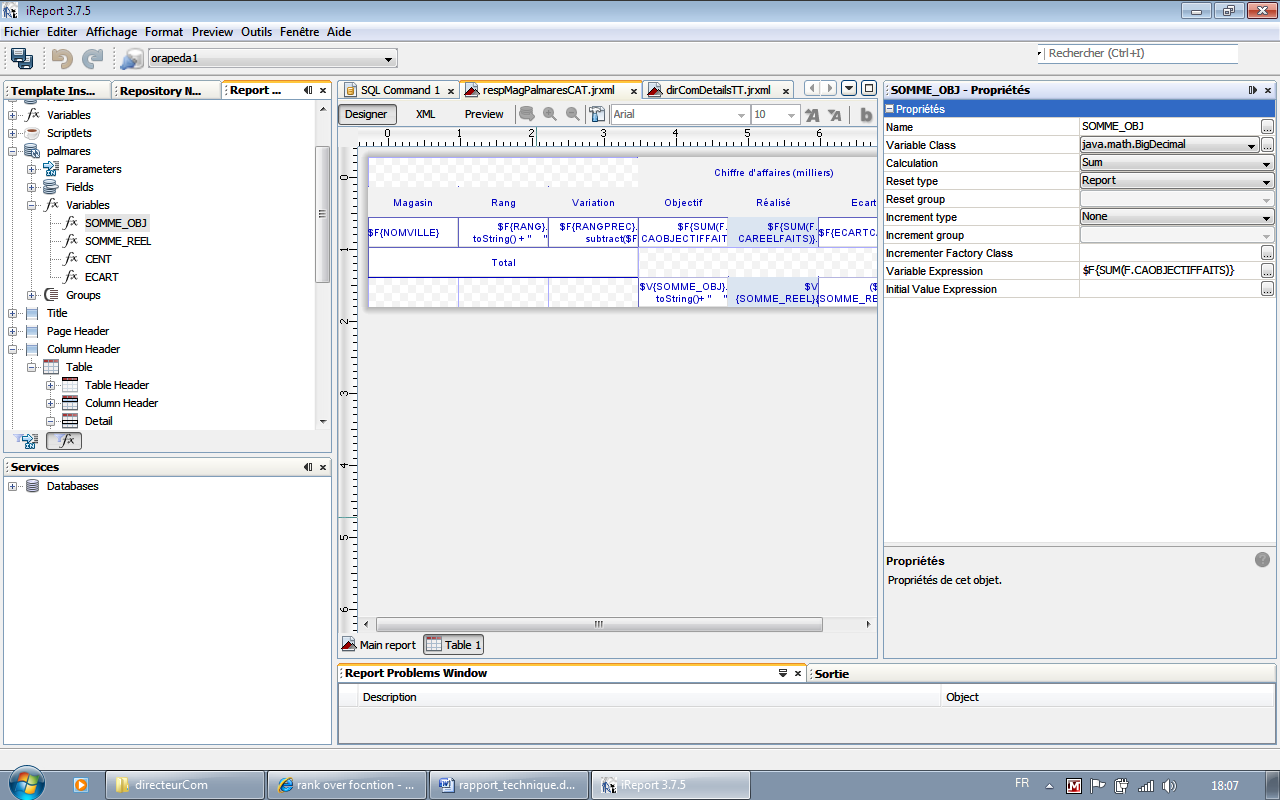


Champs saisissables

Champs provenant des *Fields* du dataset

Nous avons dû réaliser toutes les requêtes pour les affichages dans les tableaux. Pour pouvoir afficher les écarts, le calcul a été fait dans la requête SQL. De même pour le rang, il a été calculé avec l’utilisation de la clause SQL RANK OVER().

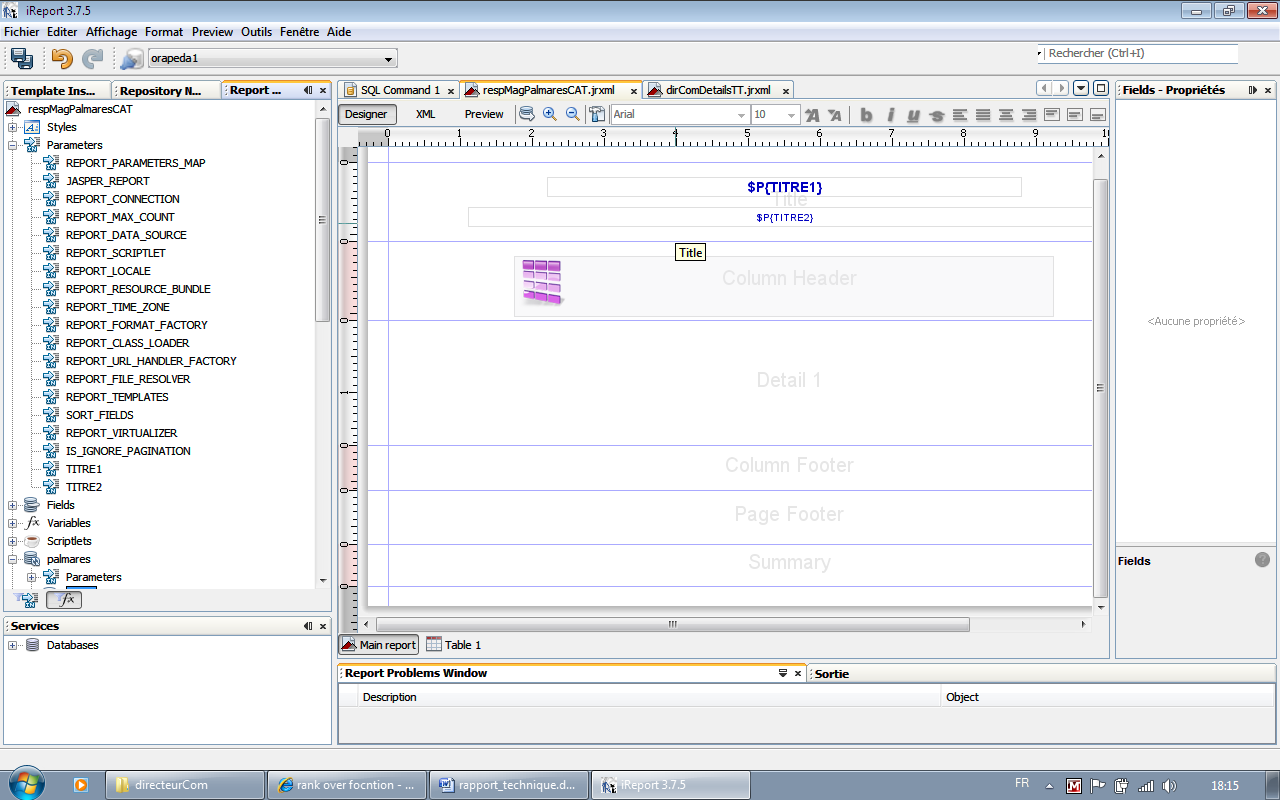
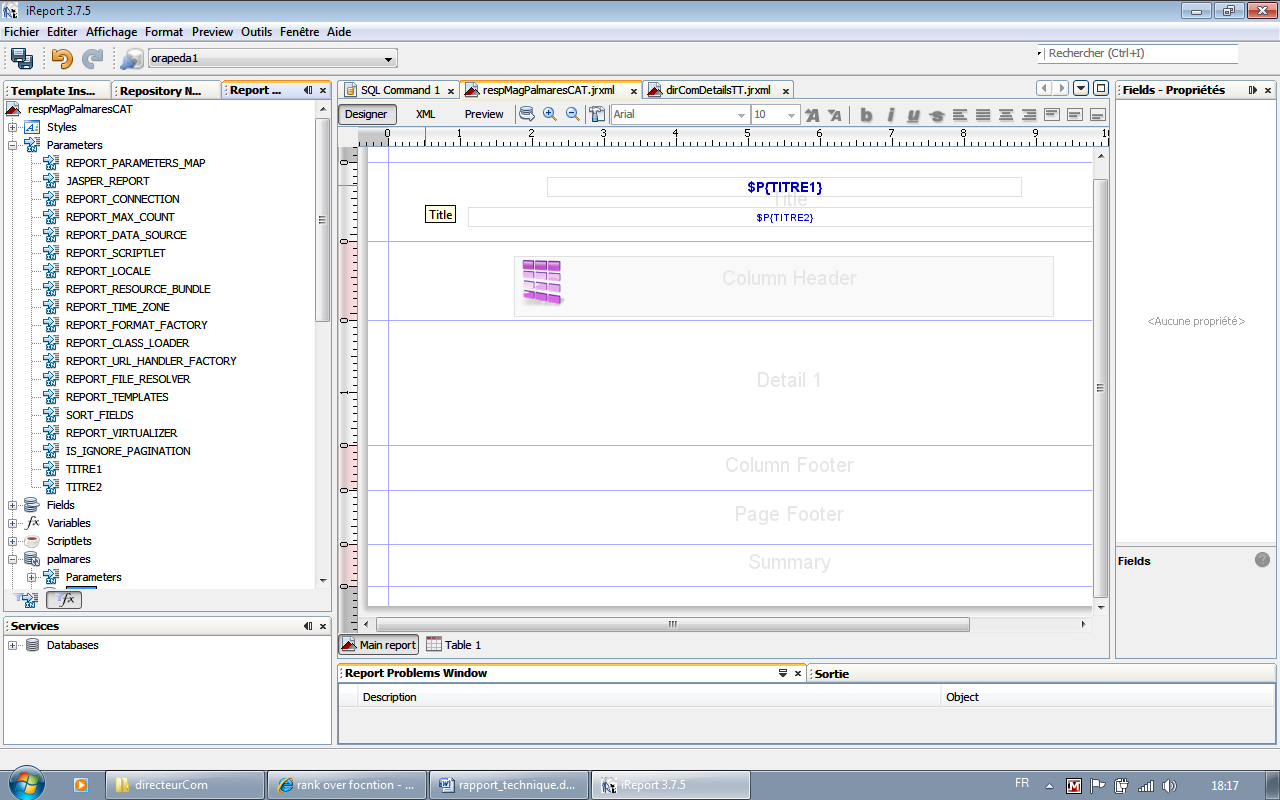
Or pour pouvoir afficher la ligne des totaux (somme correspondant au total d’une colonne et écart total), nous avons utilisé des variables. Celles-ci donnent la possibilité d’effectuer des opérations de calculs sur les expressions et de les afficher. L’opération utilisée dans ce cas est un *SUM* de l’expression à sommer. Dans l’exemple ci-dessous, on peut voir que l’on fait la somme (*calculation* *SUM*) de la variable SUM(CAOBJECTIFFAITS) *(*propriété *variable expression)*.

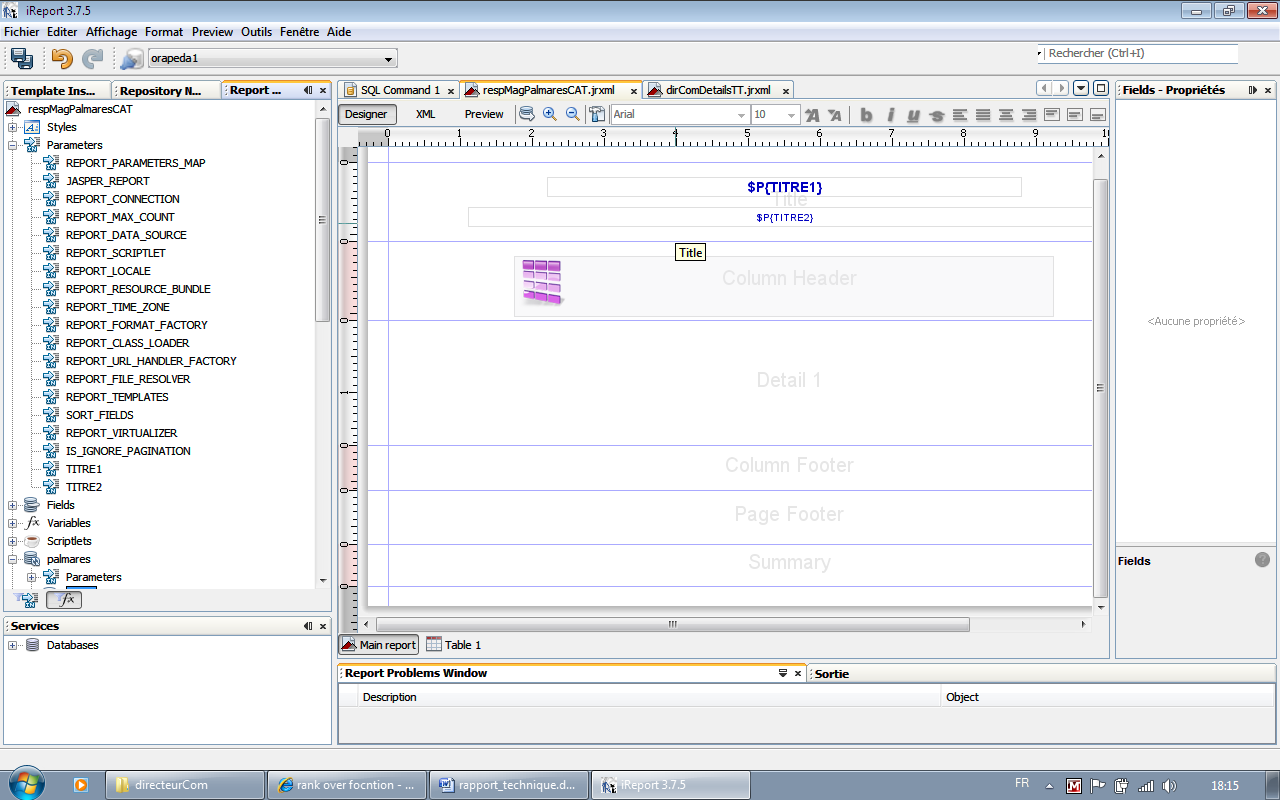


Propriété de la variable sélectionnée

Pour les écarts totaux, nous avons effectué un calcul utilisant les deux variables des totaux.

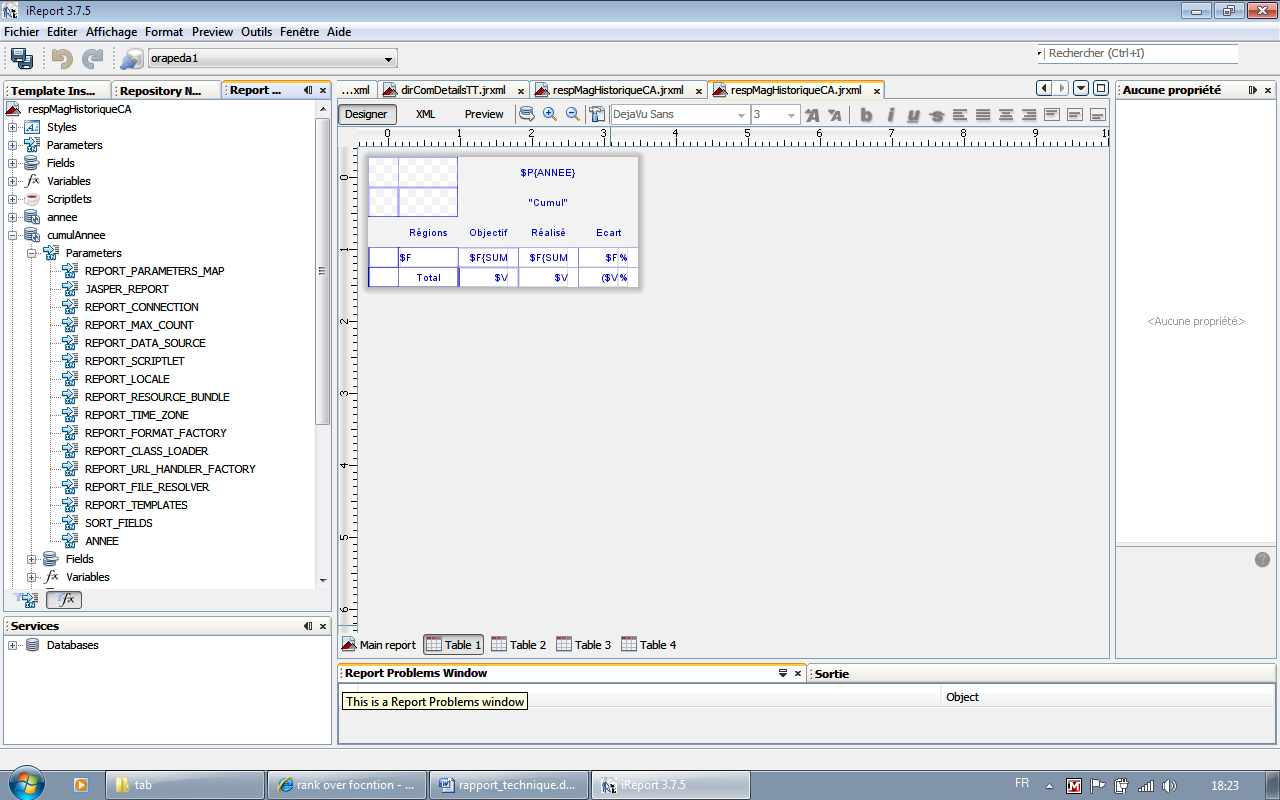
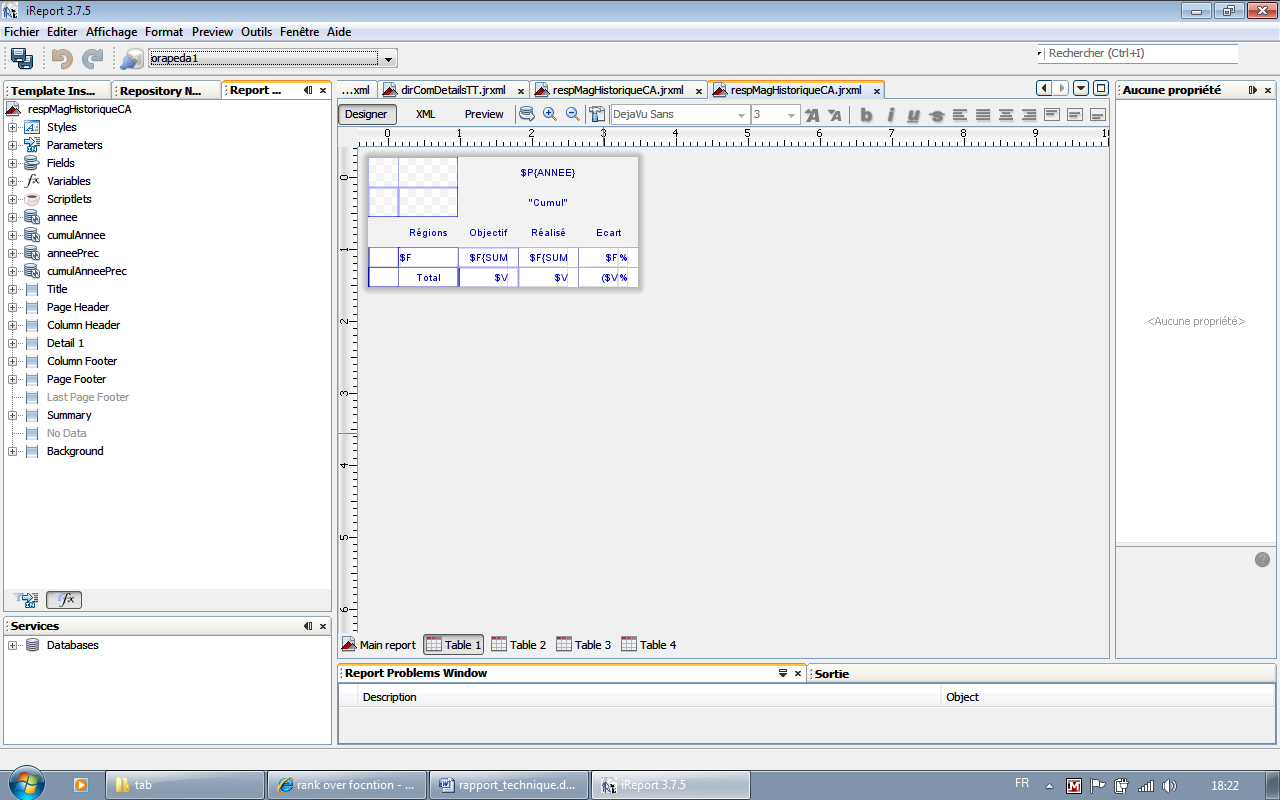
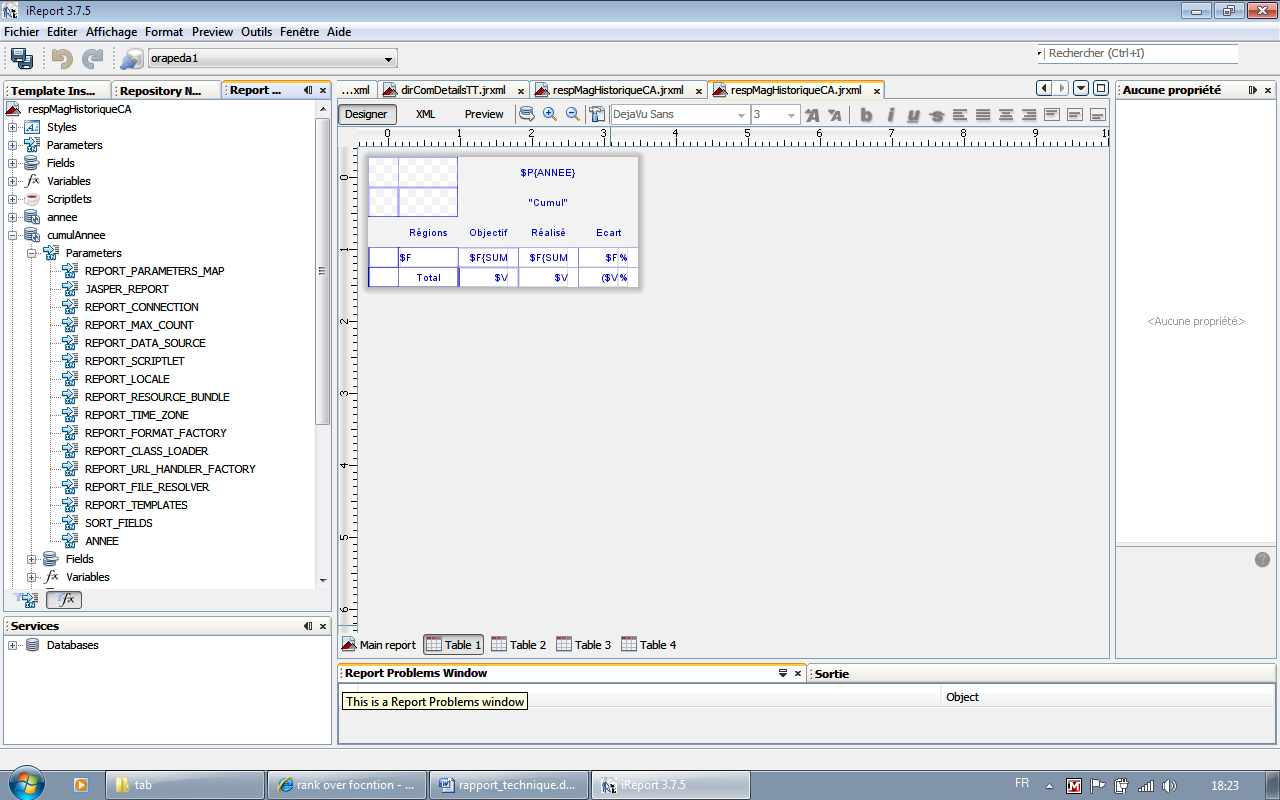
iReport permet aussi la création de paramètres. Ce sont des données prédéfinies que l’on ne peut pas trouver dans la source de données. Les paramètres sont définis pour les titres. Ces derniers seront remplacés par les choix effectués par l’utilisateur dans les filtres.





Les deux titres sont affichés dans la zone « *Title »* du rapport (voir explication partie suivante) et sont des paramètres qui prendront la valeur définie par l’utilisateur (à l’aide du code JAVA). Par exemple, si l’utilisateur veut afficher le CA, Réalisé, pour la famille Hifi, le sous-titre contiendra : « CA | Réalisé | Hifi ». Le premier titre contient le nom de l’onglet et la période choisie.

Il en est de même pour les tableaux où l’entête correspond au mois ou à l’année choisis par l’utilisateur dans les filtres « périodes ». Des paramètres ont été créés pour pouvoir être associés aux choix de l’utilisateur.



### Les différentes zones d’un rapport

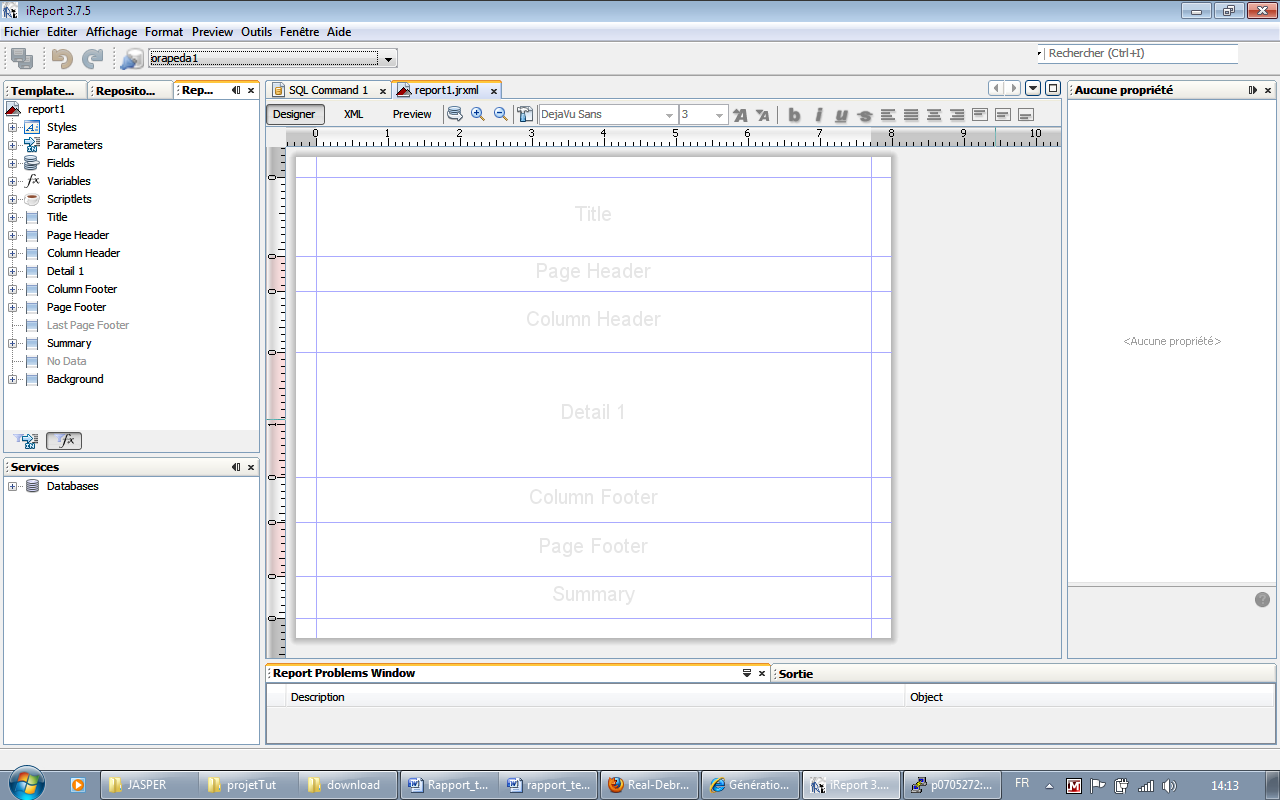
Un rapport comporte plusieurs zones. La zone *« Title » est* pour le titre et n’apparait qu’une fois en première page du rapport.

Les zones *« Page Header » et* « page Footer » correspondent à l’entête et pied de page du document. Elles apparaissent donc sur toutes les pages du rapport.

Ensuite, *« Column Header »* et *« Column Footer »* correspondent aux noms de colonnes du tableau de données et sont répétés sur chaque page.

C’est dans la zone «*Détail »* que sont affichées les lignes de résultats des colonnes que l’on désire afficher.

Enfin, la zone « *Summary* » résume des informations présentes dans la zone « *Détail* »



Pour l’application, nous avons défini le titre et sous titre de chaque tableau/graphique dans la zone *« Title ».* Les tableaux et graphiques, quant à eux sont disposés dans la zone *« Détail ».* Ils pourraient être placés dans la zone « *Column Header* » mais celle-ci est limitée en taille et cela génère une erreur lorsque le tableau contient trop de lignes. Les autres zones ne sont pas utilisées.

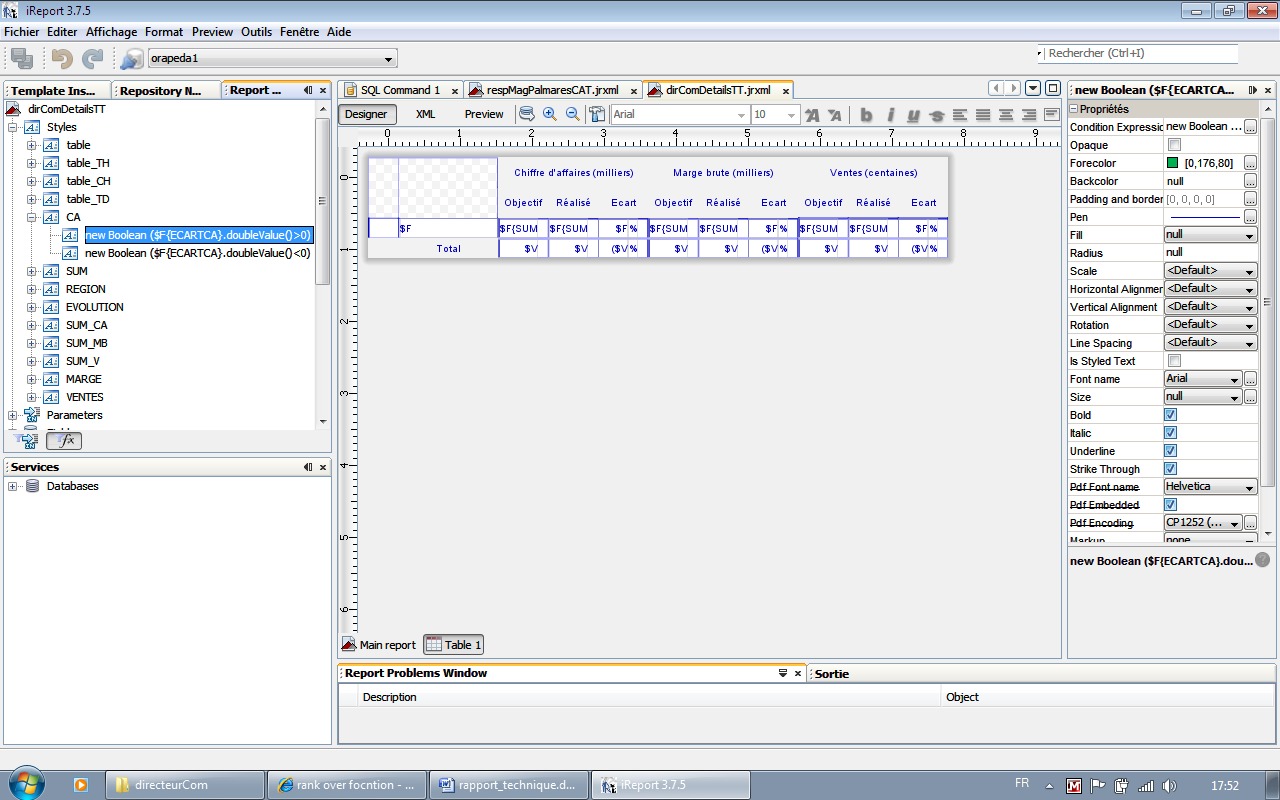
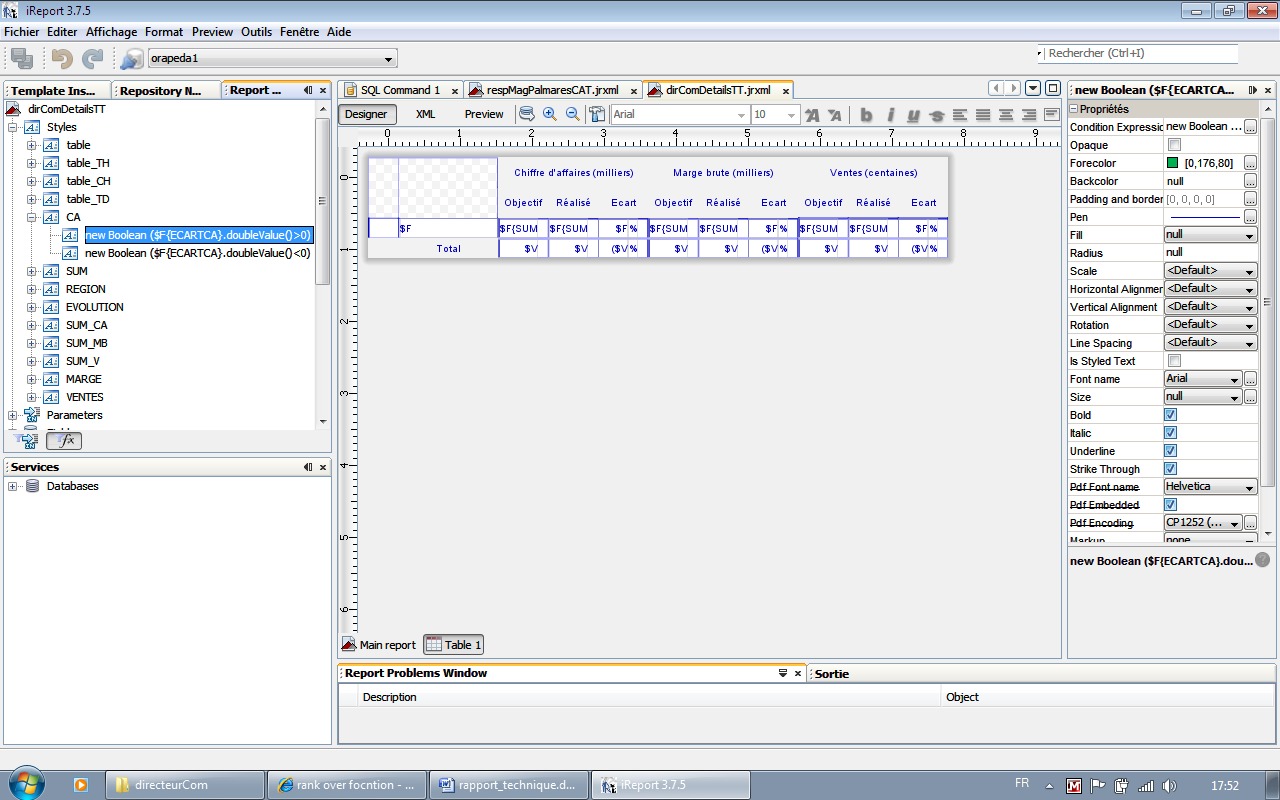
### Respect du SFD

Chaque tableau est ensuite mis aux normes du SFD avec les règles suivantes :

* La police est ARIAL de taille 10,
* Les données et les contours sont en bleu,
* Le contour du tableau, des colonnes et de certaines lignes sont deux fois plus épais que le contour standard, réservé aux lignes et aux séparations entre caractéristiques,
* Un seul chiffre après la virgule n’est autorisé pour les données,
* Les milliers sont séparés par un espace,
* Les nombres sont justifiés à droite, avec un retrait de 2 pixels, les textes sont justifiés à gauche et les titres sont centrés,
* Les cellules des entêtes de colonnes sont grisées,
* Les titres sont centrés de couleur et taille définis dans le SFD.

Pour pouvoir utiliser la couleur rouge pour les écarts négatifs et la couleur verte pour les écarts positifs, nous avons dû utiliser les styles conditionnels. Ces styles correspondent à un rapport. Nous avons tout simplement ajouté un style auquel nous avons ajouté deux conditions (nombre positif ou négatif). Ensuite, on peut modifier les propriétés de chaque condition. Par exemple, pour mettre en vert les valeurs des écarts positifs, nous avons créé un style *CA* comportant deux styles conditionnels. En première condition, si sa valeur est supérieure à 0, on lui met la couleur verte :

Propriété du style sélectionné



Couleur verte

Condition

Le second style conditionnel dans *CA* met la couleur rouge si le champ *ECARTCA* est inférieur à 0. Le style *CA* possède la couleur bleue du SFD comme couleur de texte par défaut. Ainsi, si le champ vaut 0 (aucune des deux conditions vérifiées), le texte sera bleu.

## Les graphiques

### Creation de modeles de graphiques

Un modèle de rapport se crée de la même manière que ce soit pour y afficher des résultats sous la forme de tableau ou de graphique (création d’un rapport vide puis d’un dataset).

Ensuite il faut choisir un type de graphique, dans notre cas, ce sera des camemberts ou des histogrammes.

iReport permet d’ajouter autant de séries qu’on le désire. Les histogrammes en nécessitent 2 (réalisé et objectif). Quant aux camemberts, c’est un affichage des régions. Ils nécessitent donc autant de séries que de régions, soit 5.

### Respect du SFD

Pour respecter le SFD, nous avons personnalisés les séries des graphiques, c'est-à-dire la couleur que doit représenter une donnée. La propriété « *series color* » d’un graphique permet de définir toutes les couleurs que l’on souhaite. Les légendes peuvent être affichées si le SFD le demande (généralement objectif ou réalisé ou encore le nom des villes).

# Intégration des tableaux et graphiques en Java

Une fois tous les tableaux et graphiques créés, ils doivent être intégrés dans l'interface de l'application développée en Java sous l'architecture MVC (Modèle-Vue-Contrôleur).

Ainsi, dans la partie Jasper du modèle, il existe un traitement par profil distinct (Responsable magasin, Responsable régional et Directeur commercial) ainsi qu'un traitement générique d'un document jasper. Ce dernier permet notamment le chargement d'un tableau ou d'un graphique à partir du document *jrxml*, et le traitement par profil concerne le changement de requêtes en fonction des paramètres choisis par l'utilisateur.

C'est dans la partie contrôleur que le bon fichier Jasper est choisi et appelé au chargement dans le modèle, car comme expliqué précédemment, un fichier par couple indicateur-caractéristique différent a été créé.

Enfin, la partie vue permet de faire le lien entre l'interface et les fichiers Jasper en appelant le contrôleur pour le chargement d'un nouveau fichier en transmettant tous les paramètres, chaque fois que l'utilisateur change la sélection d'une liste déroulante.

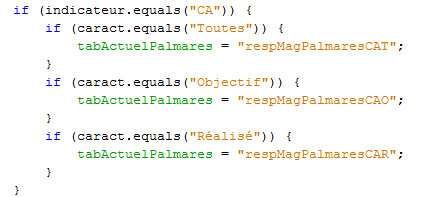
## Choix du fichier Jasper

Comme il a été rapidement énoncé en première partie, nous avons établi une règle de nommage pour les fichiers Jasper, étant donné leur grand nombre.

Ainsi, chaque nom de fichier contient le nom du profil abrégé (respMag, respReg ou dirCom) suivi du nom de l'onglet entier (Accueil, Palmares, Details, ou Historique) et se termine par les initiales du couple indicateur-caractéristique concerné (CAT, CAO, CAR, MT, MO, MR, TT, TO, TR, ...). La lettre "T" en première position signifie tout indicateur (Chiffre d'affaires, Marge et Ventes) et toutes caractéristiques en dernière position (Objectif et Réalisé).

Ensuite, un "G" est ajouté à la fin du nom de fichier lorsque ce dernier est un graphique. Chaque onglet et profil ne possède pas exactement les mêmes paramètres et tous les fichiers n'existent donc pas (par exemple les fichiers avec tous les indicateurs n'existe pas pour les onglets détails et historique).

Voici par exemple le choix du bon fichier pour le palmarès du responsable magasin si l'indicateur choisi est le chiffre d'affaires (sachant que les variables indicateur et caractéristique correspondent aux paramètres choisis et renvoyés au contrôleur par la vue) :



## Chargement et compilation d’un rapport jrxml

En ce qui concerne le chargement d'un document jrxml dans une application java, un rapport intitulé Utilisation\_Jasper\_Java a été livré en cours de projet.

Il explique le lien entre Jasper et Java à l'aide de schémas et expose le code Java nécessaire au chargement, à la compilation, au remplissage et à l'export du document initial Jasper.

Dans notre application, le chargement du tableau ou graphique est appelé dans le modèle par le contrôleur suite à la définition du nom du fichier. Ensuite, le modèle effectue un changement de requête (voir partie suivante) en fonction des paramètres choisis par l'utilisateur qui lui sont renvoyés. Puis, il compile et remplit le rapport en passant les valeurs des paramètres Jasper définis dans chaque rapport (voir partie 4). Enfin, le fichier est exporté au format HTML et affiché dans l'application par un objet navigateur. Voici une partie du code :



Le passage des paramètres sera expliqué ultérieurement (partie 4).

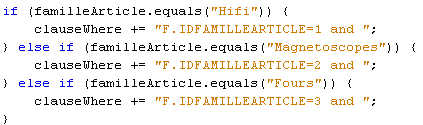
Le fichier exporté porte le même nom que le document *jrxml* chargé (variable *fic* ci-dessus), ce qui simplifie également la recherche du fichier *html* pour l'affichage. L'export en *pdf* permet de créer également le fichier *pdf* dans le cas où l'utilisateur désirerait imprimer.

## Changement de requêtes

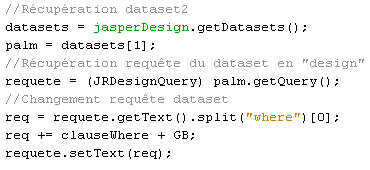
Avant la compilation d'un rapport, une méthode de changement de requête est appelée avec, en paramètres, tous les critères choisis par l'utilisateur. Cela va permettre de changer la requête du *dataset* correspondant au tableau ou graphique pour correspondre aux restrictions demandées par l'utilisateur (ce changement de requête a également été explicité en détails dans le rapport Utilisation\_Jasper\_Java, partie : Requêtes à la volée avec objets Java).

### Changement de la clause *WHERE* de la requête

Une fois les paramètres de l’utilisateur récupérés, nous pouvons remplir la clause *WHERE* de la requête. Les paramètres pris en compte dans cette clause de restriction sont : la région, la période (l’année et le mois ou cumul de mois), la famille d’articles et l’enseigne. Plusieurs tests sont alors effectués pour créer la clause, par exemple pour la famille d’article :



L’avantage de ne modifier que la clause de restriction est qu’il n’est pas nécessaire de réécrire toute la requête, celle-ci étant de taille assez conséquente. La clause *WHERE* étant la dernière clause d’une requête SQL, il suffit de récupérer la requête du *dataset* initialement créé dans le fichier Jasper, de la tronquer avant le *where* et de lui ajouter notre variable *clauseWhere*, comme suit :

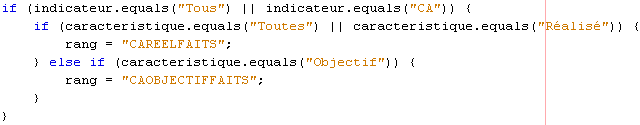


L’objet *jasperDesign*, créé précédemment au chargement du fichier *jrxml*, permet de récupérer tous les *datasets* du rapport sous forme de liste. Le *dataset* dont la requête doit être modifiée est ensuite choisi par son indice (ordre de création dans le rapport). Généralement, il y a plusieurs *datasets* s’il y a plusieurs tableaux ou graphiques dans le rapport.

La variable GB correspond à la clause *GROUPBY* qu’il faut rajouter après le *WHERE*, le groupe étant fait sur les régions ou les villes selon les cas.

### CHANGement de la clause *select* de la requete

Parfois, certains changements peuvent être également faits dans la clause de sélection de la requête, notamment le rang. En effet, comme nous l’avons décrit en première partie, le rang et la variation sont calculés dans le select de la requête. Ainsi, selon l’indicateur et la caractéristique choisis par l’utilisateur, le contenu de ce calcul va changer. Dans ce cas, nous sommes contraints de réécrire la requête entièrement car le rang intervient plusieurs fois dans la requête. Voici un exemple de test pour le chiffre d’affaires :



Lorsque tous les indicateurs sont affichés, le rang est calculé par défaut sur le chiffre d’affaires, de même que pour toutes les caractéristiques on calcule par défaut sur le réalisé.

Ensuite la requête est totalement réécrite avec les variables *rang* et *clauseWhere* puis va remplacer la requête courante du *dataset* comme vu ci-dessus.

Enfin, le contenu du *SELECT* peut aussi être modifié en fonction de si l’on veut récupérer les noms des villes ou ceux des régions, cela dépend des profils et des onglets.

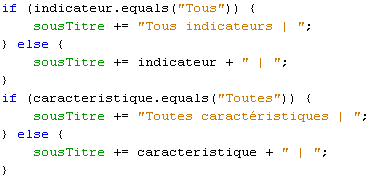
## Passage des paramètres

### Titres et sous-titres des rapports

L’utilisation de paramètres dans un rapport Jasper a été décrite dans la première partie. La plus grande utilité dans notre cas concerne l’affichage du titre et du sous-titre au dessus des tableaux. Ces titres varient selon les critères choisis par l’utilisateur, ils doivent donc être transmis au rapport avant la compilation.

Ces titres sont déjà définis dans le rapport à la bonne position et le style correspond au SFD.

Voici un exemple de remplissage du sous-titre pour la partie caractéristique :



Ensuite, tous les paramètres sont ajoutés dans une table *HashMap* et transmis au remplissage du tableau (voir variable *parameters* dans la capture d’écran du *IV. 2.*) :



### Periodes dans l’entete des tableaux

Les paramètres ont également été utilisés pour les onglets *Historique* et *Détails*. En effet, ces tableaux affichent en entête le nom du mois ou du cumul (janvier à mars par exemple) concerné. Cela dépend donc de la période choisie par l’utilisateur.

Ces paramètres sont définis dans un test sur la période sélectionnée et envoyés au moment du remplissage du rapport de la même manière que précédemment.

Enfin, pour le profil « Responsable régional », l’onglet Historique affiche 12 petits tableaux par mois affichant les valeurs cumulées de janvier au mois de chaque petit tableau.

Or, dans ce cas, l’utilisateur choisit une année et le mois jusqu’auquel il veut afficher les tableaux. Pour faire cela, nous avons donc réalisé 12 tableaux distincts sous Jasper que nous avons collés les uns aux autres. Ainsi, nous avons pu utiliser les styles pour conditionner l’affichage de chaque petit tableau à la valeur du paramètre (non vide).

Nous avons donc utilisé 12 paramètres ayant une double fonction : l’affichage du nom du mois dans l’entête de chaque petit tableau et la mise en condition de l’affichage du tableau.

# Conclusion

Le développement de cette application BI fut très enrichissant et nous a permis de mettre en pratique les notions théoriques sur l’aide à la décision acquises tout au long de notre formation.

Nous avons du faire plusieurs choix au cours du développement, notamment sur l’outil de reporting au début et sur la manière la plus adaptée de l’utiliser pour notre application.

Nous ne regrettons pas le choix de l’outil Jasper pour sa facilité de mise en place sa bonne adéquation avec le rendu désiré dans l’application. En effet, la majorité des tableaux et graphiques ont pu être réalisés en respectant scrupuleusement le SFD. Cet outil demande une certaine prise en main au début mais permet ensuite une utilisation assez pratique et rapide.

Nous avons également fait le choix de créer beaucoup de fichiers Jasper différents et d’effectuer ainsi moins de traitements dans le code Java. Nous sommes plutôt satisfaits de ce choix en fin de développement, car malgré le fait que la création des tableaux fut fastidieuse, le changement de tableaux demande un temps relativement acceptable. D’autant plus qu’il n’était pas possible de changer aisément la structure d’un tableau Jasper en Java (changement du nombre de colonnes).

Pour conclure, nous avons apprécié de parvenir au développement total de l’application pour la partie Jasper et avons beaucoup appris dans ce projet qui nous a également permis d’évaluer notre autonomie.