JasperReports & iReport

Guide d'utilisation

Version 0.1 du 15/09/2008

Etat : Rédaction

SUIVI DES MODIFICATIONS

Version	Rédaction	Description	Vérification	Date
0.1	JP.Wilsch	Création		15/09/08
		Document validé dans sa version xxx		

LISTE DE DIFFUSION

Organisation	Nom	Info	Commentaire	Validation
		☐	☐	☐
		☐	☐	☐
		☐	☐	☐

SOMMAIRE

DOCUMENTS DE REFERENCE

Version	Titre

1 PRESENTATION DE JASPERREPORTS

1.1 QU'EST CE QUE JASPERREPORTS ?

JasperReports est une librairie Java open source dédiée à l'ajout de capacités de reporting aux applications Java, Web ou stand alone.

Démarré en 2001 par Teodor Danciu, le projet est aujourd'hui porté par la société JasperSoft.

JasperReports permet la représentation de données sous forme textuelle, mais aussi la génération de graphiques divers (sous forme de camembert, barre, courbe, nuage de point).

Les fonctionnalités principales de JasperReports sont :

- Une structure de page flexible
- Possibilité de présenter les données de manière variée (textuel, graphique)
- Possibilité de fournir les données sous différentes formes (paramètres, sources de données)
- Gestion de sous rapports
- Export dans une grande variété de formats

1.2 STRUCTURE DE PAGE FLEXIBLE

JasperReports permet de séparer les données du rapport en différentes sections :

- Le titre, qui apparaît une fois, au début du rapport
- L'entête de page, qui apparaît au début de chaque page
- Le détail, qui contient habituellement les principales données du rapport
- Le pied de page, qui apparaît à la fin de chaque page
- Le résumé, qui apparaît une fois, à la fin du rapport

JasperReports permet le contrôle dynamique de cette structure en fonction du contenu du rapport, en laissant la possibilité par exemple d'afficher ou non des données en fonction de paramètres du rapport.

Les données peuvent également être rassemblées dans des sections logiques, appelées groupes, en fonction de leurs valeurs. La définition de groupes de données peut également servir à calculer des moyennes ou pourcentages sur ces données.

1.3 PLUSIEURS FACONS DE PRESENTER LES DONNEES

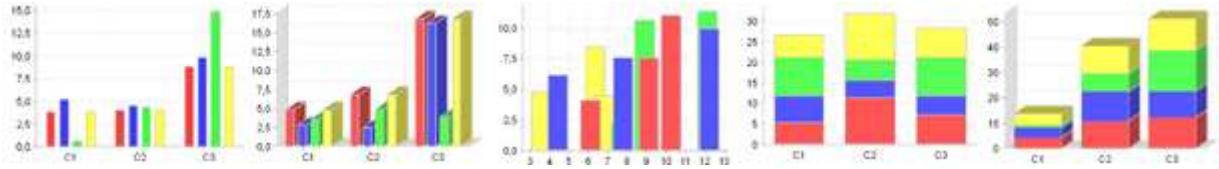
JasperReports propose de présenter les données sous forme textuelle, ou graphiquement sous forme de diagrammes.

Ces diagrammes vont permettre d'afficher des données dynamiques, qui ne seront pas passées directement au rapport mais calculées à partir des données fournies.

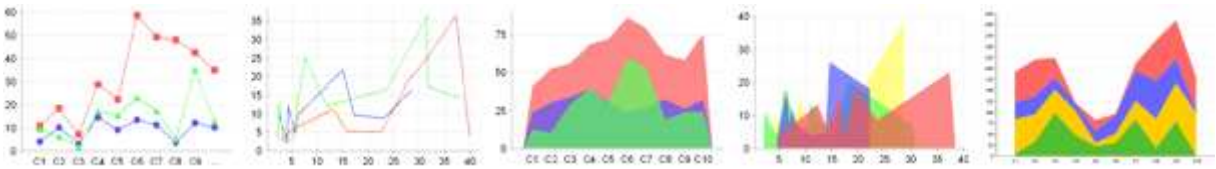
La palette de représentation graphique proposée est large :

- le camembert (2D et 3D)

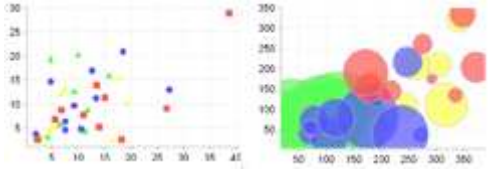
- les barres (2D, 3D, en relief, empilées 2D, empilées 3D)



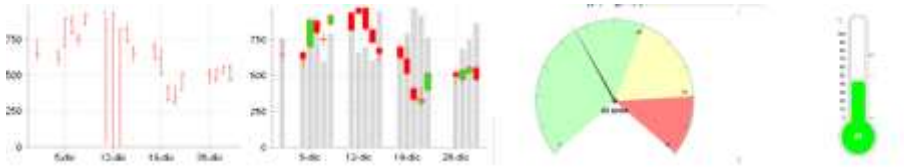
- les courbes



- les nuages de points



- minimum/maximum, cadran, thermomètre, …



Dès éléments graphiques de bases peuvent également très facilement être ajoutés aux rapports (image, ligne, rectangle, ellipse, …) pour l'agrémenter.

1.4 PLUSIEURS FAÇONS DE FOURNIR LES DONNEES

JasperReports autorise le développeur à fournir les données au rapport sous la forme de paramètres. Ces paramètres peuvent être des instances de n'importe quelle classe Java.

Les données peuvent être récupérées dans une de base de données, par l'intermédiaire d'une connexion JDBC fournie au rapport. Les requêtes de récupération des données seront inscrites dans le rapport.

Enfin, les données peuvent aussi être fournies en utilisant des classes spécifiques de la librairie appelées datasources (sources de données), implémentant une interface commune. JasperReports inclus un certain nombre de datasource, permettant d'utiliser comme source de données pour le rapport :

- Des resultSet JDBC
- Des objets de type Map
- Des beans Java
- Des fichiers XML
- …

Il est également aisé, en cas de besoin spécifique, de réaliser sa propre datasource, en implémentant l'interface commune à toutes ces classes.

1.5 SOUS RAPPORTS

Une autre fonctionnalité de JasperReports est la possibilité d'utiliser des sous rapports, et de créer un rapport à partir de plusieurs autre rapports. Tout rapport peut devenir le sous rapport d'un autre.

Cette technique propose deux avantages principaux :

- Simplification de la réalisation d'un rapport en découpant la complexité (plusieurs petits rapports simples pour faire un rapport complexe)
- Possibilité de réutilisation dans plusieurs rapports différents d'un même sous rapport présentant un affichage fréquent (cet affichage n'a donc besoin de n'être développé qu'une fois)

1.6 EXPORT

JasperReports propose l'export de ces rapports dans de nombreux formats :

- PDF
- XLS (Excel)
- RTF
- ODF
- HTML

- XML
- CSV
- Texte brut

Pour tous ces formats, la librairie propose l'exportation directement sous la forme d'un flux de données, qui pourra par exemple être directement affiché dans un navigateur Internet par exemple, ou l'enregistrement dans un fichier.

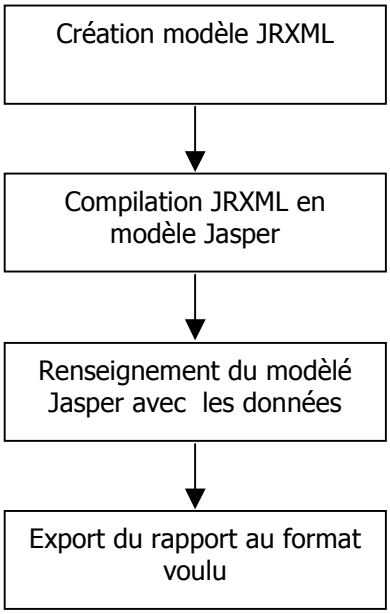
1.7 DEPENDANCES

JasperReports s'appuie sur plusieurs librairies Java open source pour implémenter ses fonctionnalités, parmi lesquelles :

- iText pour la génération de PDF
- JFreeChart pour la génération de diagrammes et graphiques
- Jakarta POI pour la génération de documents Microsoft Office
- JAXP pour le parsing et la transformation de XML

1.8 UTILISATION TYPIQUE

Le schéma suivant illustre le déroulement d'une utilisation typique de JasperReports :



Un rapport est décrit dans un modèle de rapport au format XML. Ces modèles peuvent être écrits à la main ou générer par des outils graphiques comme iReport, qui sera présenter par la suite. Ces fichiers XML portent l'extension .jrxml.

Ces modèles au format XML sont ensuite compilés dans un format binaire. Cette compilation peut ce faire soit programmatiquement via une API, soit via une tache ANT. Le résultat de cette compilation est un fichier Jasper qui porte l'extension .jasper.

Ce rapport compilé est ensuite renseigné avec les données à afficher (le terme *filled* est utiliser en anglais dans la documentation). Ce rapport renseigné est appelé impression Jasper et peut éventuellement être sauvé tel quel dans un fichier à l'extension .jrprint.

Cette impression Jasper sera plus certainement ensuite exportée dans un des formats proposés par JasperReports.

1.9 FONCTIONNALITES AVANCEES

JasperReports propose également pour la réalisation de rapports un lot de fonctions avancées telles que :

- Gestion de l'internationalisation
- Scriptlets (portions de code Java pouvant être exécuté lors du renseignement du rapport)
- Tableaux croisés
- Ancres et liens dans les documents
- Gestion des marques pages dans les fichiers PDF
- Gestion des formules dans les documents Excel
- Gestion avancée de la mémoire pour les très gros rapports (Virtualizer)

2 PRESENTATION DE IREPORT

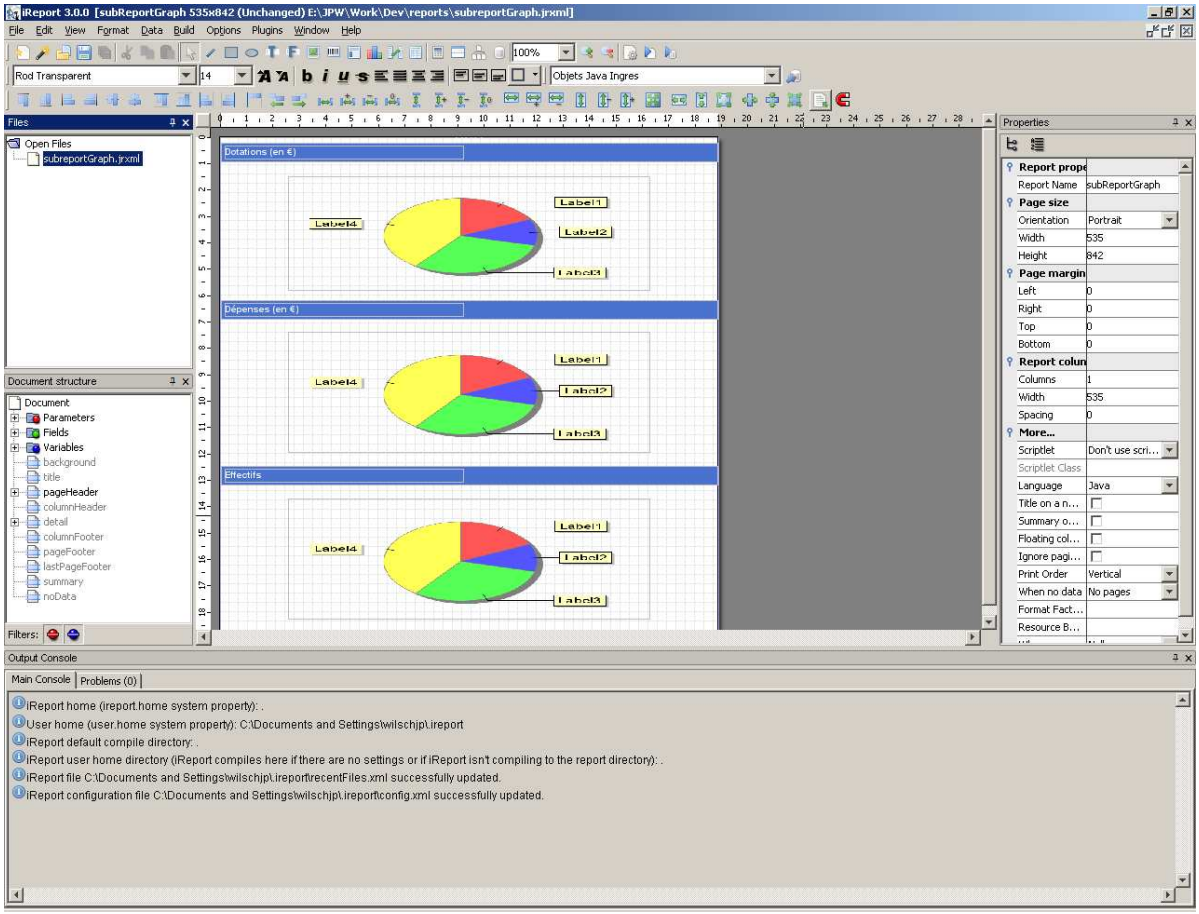
2.1 QU'EST CE QUE IREPORT ?

IReport est un logiciel open source, écrit entièrement en Java, permettant, par l'intermédiaire d'une interface graphique riche, de créer des modèles de rapports au format jrxml de JasperReports. L'utilisation de ce logiciel permet de s'abstraire de la complexité de la syntaxe XML de JasperReports, et de gagner du temps lors du développement de modèles de rapport.

IReport permet une prise en main complète de JasperReports via son interface graphique, par son support complet des tags XML de la librairie, une interface WYSIWYG pour tous les éléments graphiques, un éditeur d'expressions, la gestion des sous rapports.

Un module intégré d'exportation, associé à un support des connexions JDBC et des datasources JasperReports, permet également de tester le rendu des rapports directement depuis le logiciel.

IReport apporte à JasperReports un gain de productivité non négligeable, une fois l'outil pris en main, par rapport à d'autres solutions de reporting pour Java non outillées.



IReport n'est pas le seul outil d'édition existant pour JasperReport. On peut citer les logiciels suivants :

- JasperAssistant (plugin Eclipse, payant)
- JasperPal (stand-alone, gratuit)

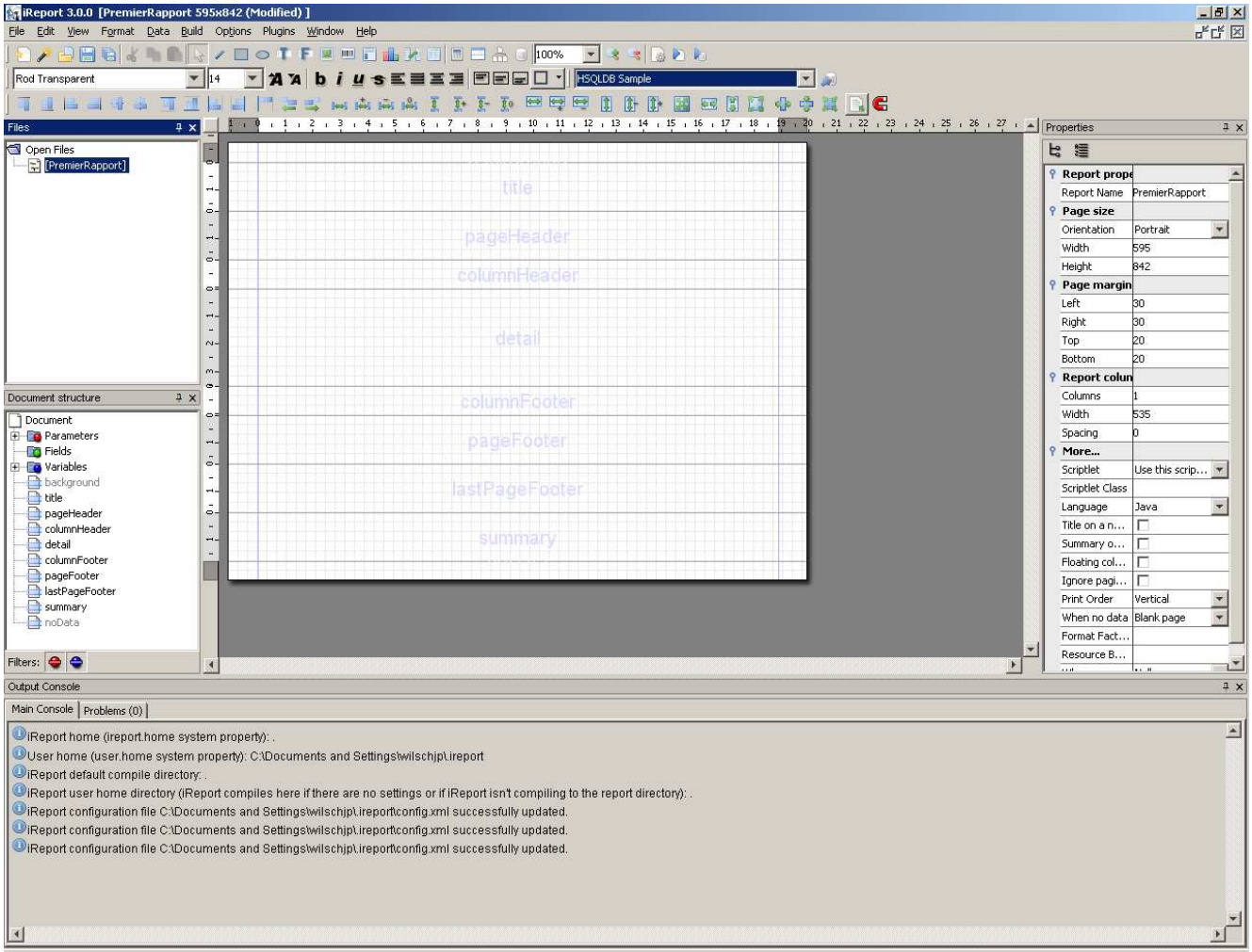
IReport est néanmoins l'éditeur officiel de JasperReports, réalisé et supporté par la même équipe que la librairie de reporting (iReport était à la base un projet indépendant, avant de voir sa popularité reconnaître par JasperSoft et l'auteur du logiciel embauché par la société).

3 CREATION D'UN PREMIER RAPPORT

3.1 CREATION DU MODELE DE RAPPORT JRXML

La première étape de la création d'un rapport est la création du modèle de rapport JRXML correspondant. L'outil iReport sera utilisé pour réaliser ce modèle.

Un clic sur l'icône Nouveau Document  de iReport ouvre un nouveau modèle vierge :



Ce modèle est un fichier XML, dont voici le code source juste après son initialisation par iReport :

```
<?xml version="1.0" encoding="UTF-8"  ?>
<!-- Created with iReport - A designer for JasperReports -->
<!DOCTYPE jasperReport PUBLIC "//JasperReports//DTD Report Design//EN"
"http://jasperreports.sourceforge.net/dtds/jasperreport.dtd">
<jasperReport
                name="PremierRapport"
                columnCount="1"
                printOrder="Vertical"
                orientation="Portrait"
                pageWidth="595"
                pageHeight="842"
                columnWidth="535"
                columnSpacing="0"
                leftMargin="30"
                rightMargin="30"
                topMargin="20"
                bottomMargin="20"
                whenNoDataType="BlankPage"
                isTitleNewPage="false"
                isSummaryNewPage="false">
        <property name="ireport.scriptlethandling" value="2" />
        <property name="ireport.encoding" value="UTF-8" />
        <import value="java.util.*" />
        <import value="net.sf.jasperreports.engine.*" />
        <import value="net.sf.jasperreports.engine.data.*" />


                <background>
                        <band height="0"  isSplitAllowed="true" >
                        </band>
                </background>
                <title>
                        <band height="50"  isSplitAllowed="true" >
                        </band>
                </title>
                <pageHeader>
                        <band height="50"  isSplitAllowed="true" >
                        </band>
                </pageHeader>
                <columnHeader>
                        <band height="30"  isSplitAllowed="true" >
                        </band>
                </columnHeader>
                <detail>
                        <band height="100"  isSplitAllowed="true" >
                        </band>
                </detail>
                <columnFooter>
```

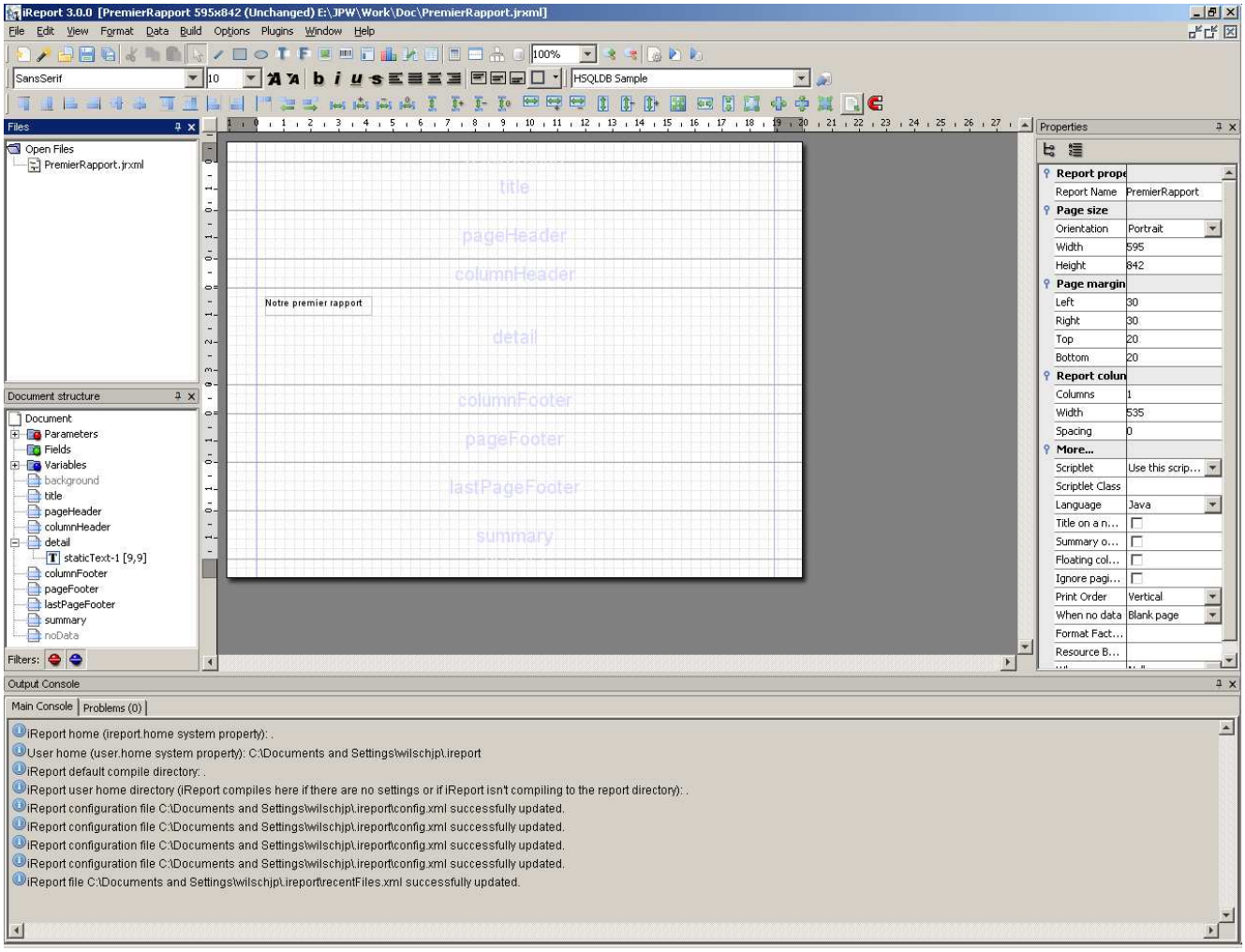
```
                        <band height="30"  isSplitAllowed="true" >
                        </band>
                </columnFooter>
                <pageFooter>
                        <band height="50"  isSplitAllowed="true" >
                        </band>
                </pageFooter>
                <lastPageFooter>
                        <band height="50"  isSplitAllowed="true" >
                        </band>
                </lastPageFooter>
                <summary>
                        <band height="50"  isSplitAllowed="true" >
                        </band>
                </summary>
</jasperReport>
```

On y remarque le tag englobant de tout modèle de rapport, `jasperReport`, et les tags des différents éléments constituant un modèle de rapport :

- `background` : pour définir la fond des pages du rapport, peut être une image, du texte, ou un élément du type watermark.
- `title` : pour définir le titre du rapport, qui apparaît une seule fois au début du rapport
- `pageHeader` : pour définir l'entête de page, qui apparaît au début de chaque page
- `columnHeader` : pour définir l'entête de colonne
- `detail` : pour définir le détail du contenu du rapport, le contenu de cet élément est répété pour chaque enregistrement de la datasource du rapport
- `columnFooter` : pour définir le pied de colonne
- `pageFooter` : pour définir le pied de page, qui apparaît à la fin de chaque page
- `lastPageFooter` : pour définir le pied de page de la dernière page du rapport, à la place du pied de page définit dans l'élément `pageFooter`
- `summary` : pour définir le résumé du rapport, qui apparaît une seule fois à la fin du rapport

Dans ce premier rapport, nous allons afficher un texte statique (c'est-à-dire non dynamique).

Pour cela, on sélectionne l'outil Texte Statique de iReport  et on dessine dans la section `detail` du rapport un rectangle représentant notre zone de texte. Le texte sera ensuite modifiable en double cliquant sur l'élément.



Le code source de notre section detail a maintenant la forme suivante :

```
<detail>
        <band height="100"  isSplitAllowed="true" >
                <staticText>
                        <reportElement x="9" y="9" width="111"
                                        height="20" key="staticText-1"/>
                        <box></box>
                        <textElement>
                                <font/>
                        </textElement>
                        <text><![CDATA[Notre premier rapport]]></text>
                </staticText>
        </band>
</detail>
```

On remarque les éléments :

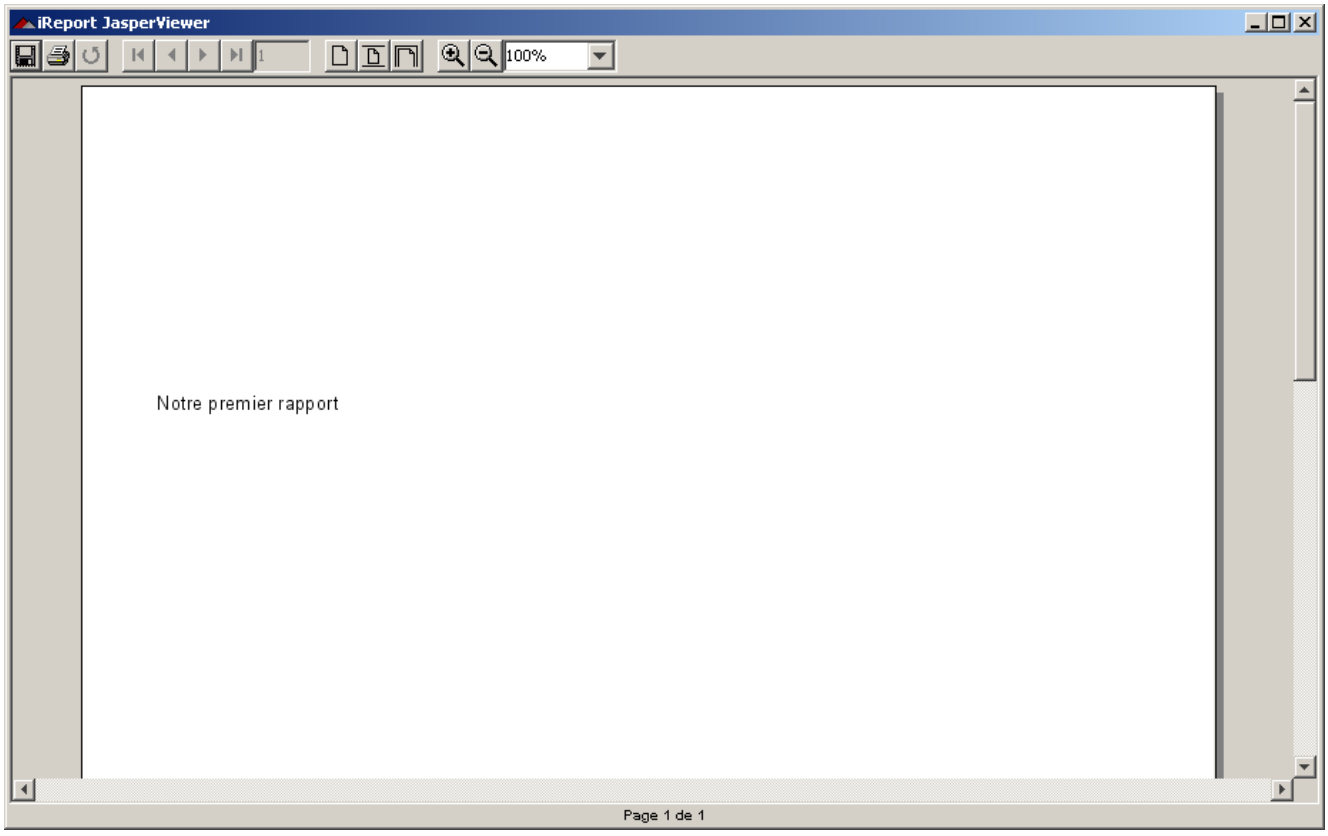
- `staticText` : pour définir un texte statique, qui ne dépend ainsi d'aucune datasource, variable, paramètre ou expression
- `reportElement` : pour définir la position et la taille d'un élément.
- `text` : pour définir le texte affiché par la zone de texte statique

3.2 PREVISUALISATION D'UN MODELE DE RAPPORT

IReport intègre la fonctionnalité de prévisualisation du modèle de rapport en cours de création. Le menu Créer offre plusieurs possibilités, notamment :

- Compiler : pour tester la bonne compilation de notre modèle de rapport
- Exécuter (avec une source de données vide ou non) : pour visualiser notre rapport, avec le choix du type de datasource que l'on lui passe (nous reviendrons sur les datasources dans le chapitre suivant)
- Choix du type d'exportation : pour choisir, parmi les types d'exportation offerts par JasperReports, celui avec lequel on souhaite prévisualiser le rapport : PDF, HTML, Excel, CSV ou le visualiseur intégré à JasperReports, JRViewer.

Voila à quoi ressemble notre rapport avec JRViewer :



3.3 COMPILATION VIA ANT

Pour pouvoir être visualiser, le modèle de rapport doit donc d'abord passer par une étape de compilation durant laquelle il passe du format jrxml au format binaire, que iReport réalise automatiquement lors de l'utilisation de sa fonctionnalité de prévisualisation.

Pour une utilisation de JasperReports en mode projet, lors duquel un nombre important de modèle de rapport peuvent être écrit, JasperReports propose une tâche ANT permettant la compilation en masse de modèles de rapport. Cette tache se nomme JRC, défini dans la classe `net.sf.jasperreports.ant.JRAntCompileTask`. Elle s'utilise de la manière suivante :

```
<project name="Jasper" default="compile">
	<description>Jasper</description>
	<path id="classpath">
		<fileset dir="./">
			<include name="**/*.jar" />
		</fileset>
	</path>
	<target name="compile" description="Compile JRXML reports">
		<taskdef name="jrc"
classname="net.sf.jasperreports.ant.JRAntCompileTask">
			<classpath refid="classpath" />
		</taskdef>
		<jrc destdir="./bin/report/jasper">
			<src>
				<fileset dir="./src/report/jasper">
					<include name="**/*.jrxml" />
				</fileset>
			</src>
			<classpath refid="classpath" />
		</jrc>
	</target>
</project>
```

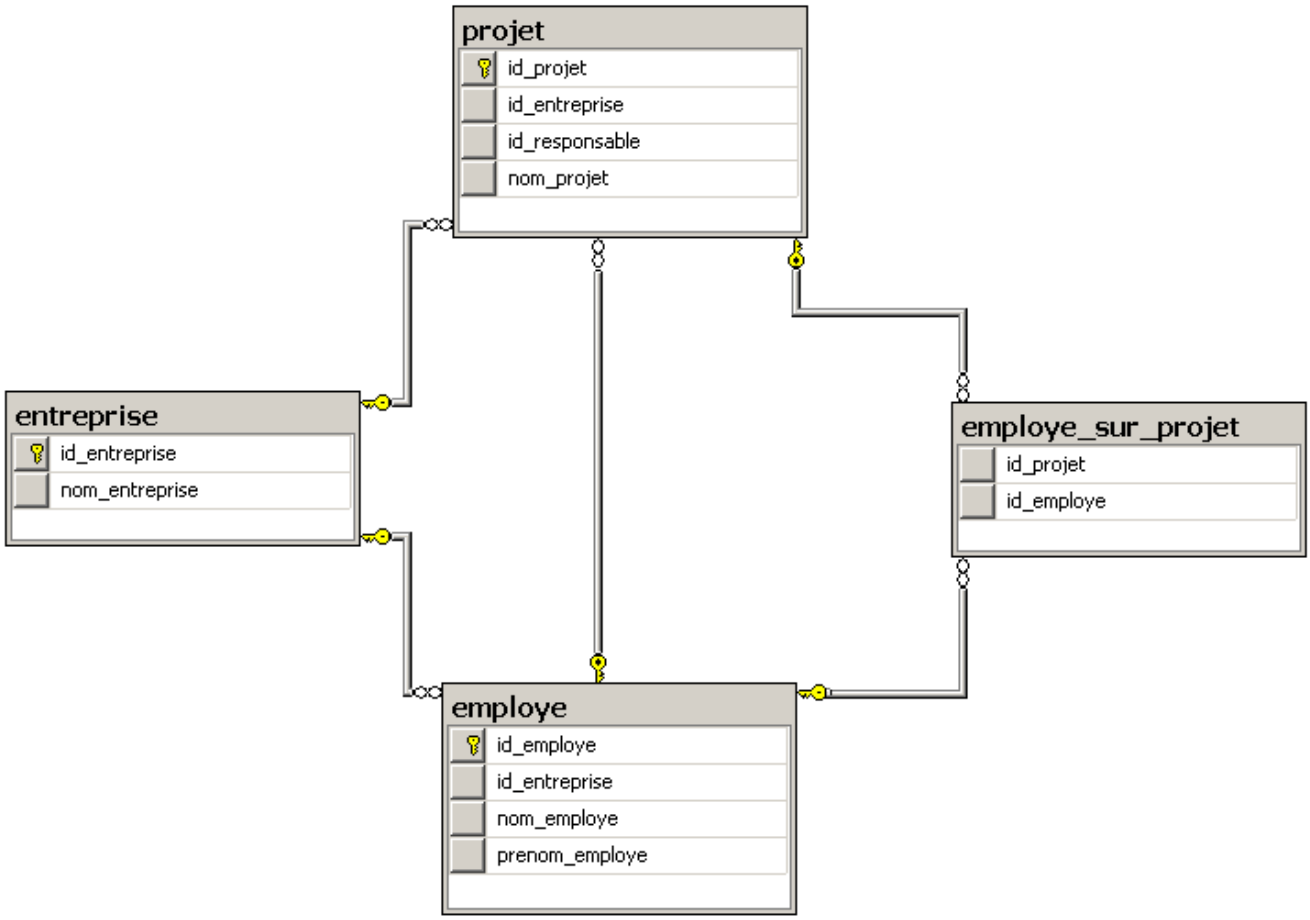
La tâche JRC nous permet d'indiquer le répertoire de destination des modèles de rapport compilés, et la source de ces modèles de rapport, ici sous la forme d'un fileset.

4 CREATION D'UN RAPPORT DYNAMIQUE

Nous avons vu comment créer un premier rapport, ne contenant que des données statiques. Nous allons voir dans ce chapitre comment créer un rapport contenant des données issues d'une base de données, d'objets Java ou d'un fichier XML.

4.1 RAPPORT ASSOCIE A UNE BASE DE DONNEES

Dans le cadre de ce guide, nous utiliserons une base de données sous SQLServer, ayant la structure suivante :



Nous allons commencer par créer un rapport affichant la liste des projets contenus dans notre base. Le rapport indiquera le nom du projet, de l'entreprise dans laquelle il se déroule, et le nom et le prénom de son responsable.

La requête suivante nous donnera ses informations :

```
SELECT
        p.nom_projet,
```

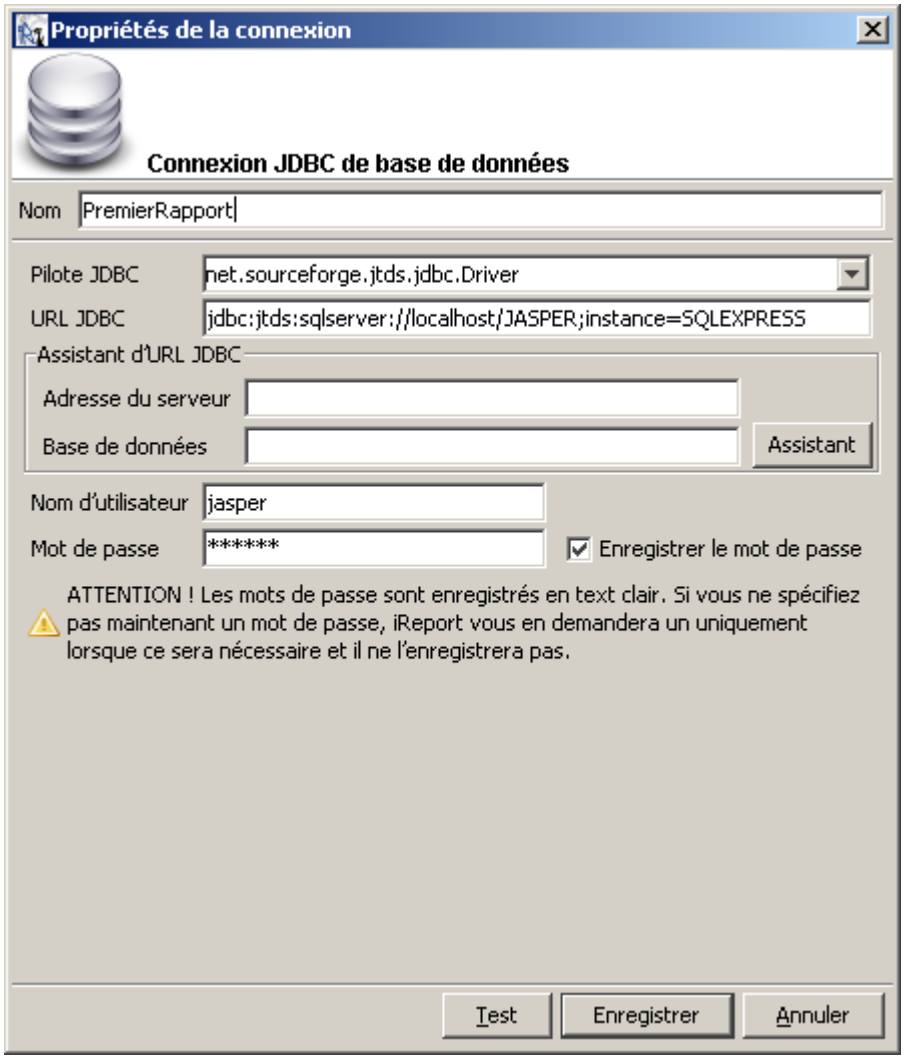
```
        e.nom_entreprise,
        em.nom_employe,
        em.prenom_employe
FROM
        projet p,
        entreprise e,
        employe em
WHERE
        p.id_entreprise = e.id_entreprise AND
        p.id_responsable = em.id_employe
```

Nous allons maintenant intégrer cette requête SQL dans notre modèle de rapport.

4.1.1 CONFIGURATION DE LA CONNEXION

Avant cela, nous allons configurer iReport afin qu'il se connecte à notre base de données, en passant par le menu Données > Connexions/Sources de données, puis le bouton Nouveau, et enfin en choisissant Connexion JDBC dans la liste, pour créer une nouvelle connexion à notre base.

Une assistant nous permet de saisir le type de driver JDBC que nous souhaitons utiliser, et les informations de connexion à notre base.

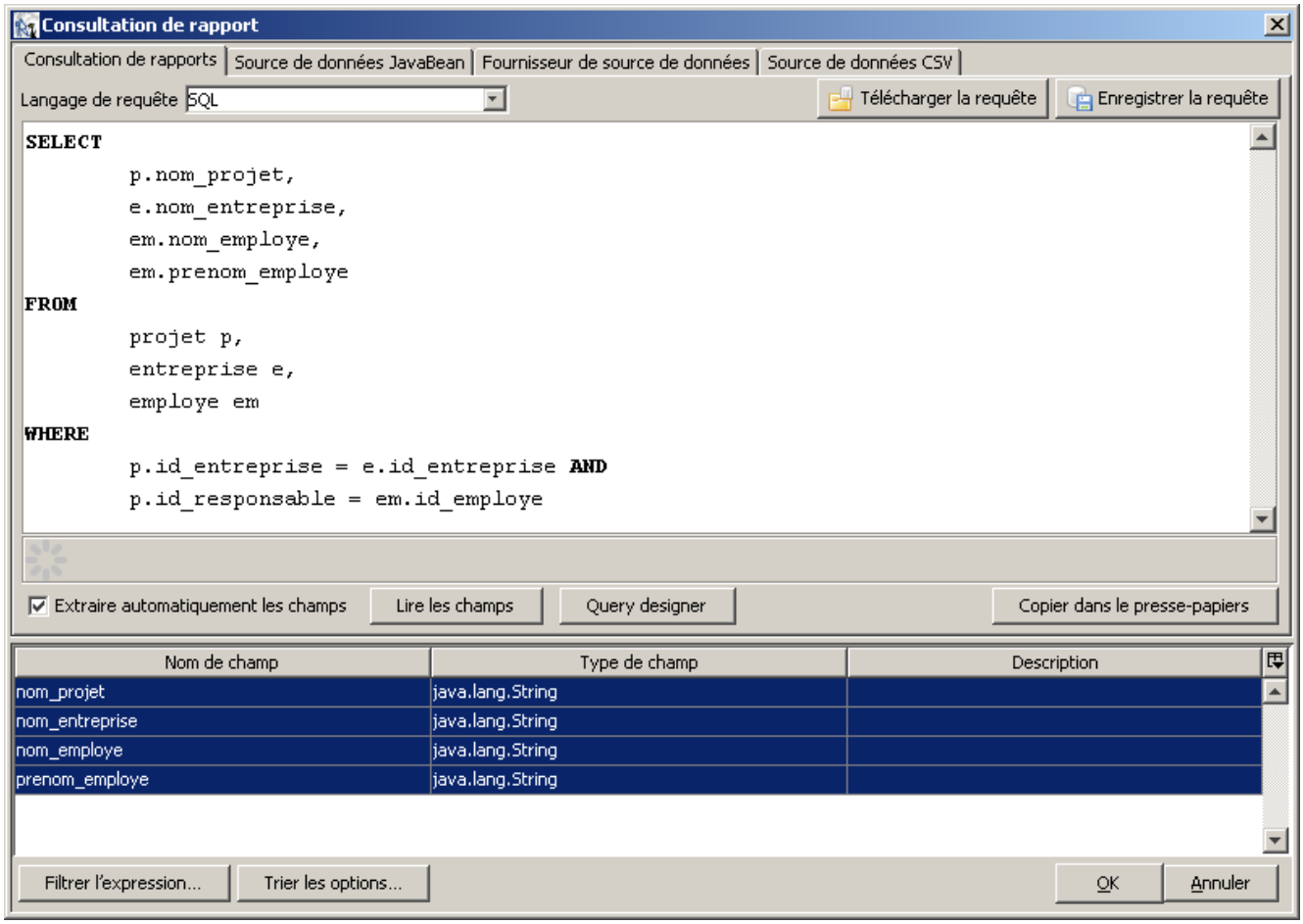


Une fois cette connexion définie, elle sera utilisée par iReport comme connexion pour notre rapport afin d'accéder aux données.

Dans le cadre de la génération d'un rapport par une application Java, cette connexion JDBC sera créée dans le code puis passée en paramètre à la méthode de remplissage du rapport.

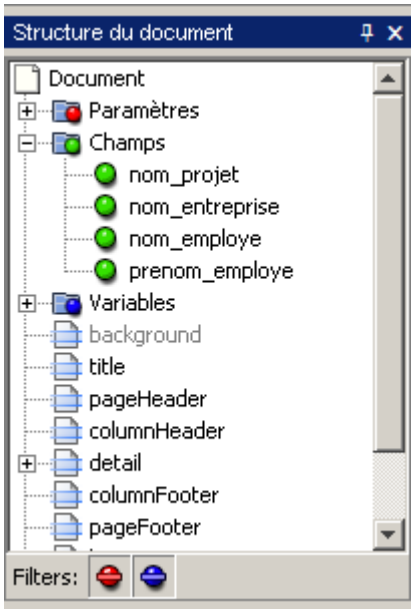
4.1.2 INSCRIPTION DE LA REQUETE

La requête de récupération des données va pouvoir être inscrite dans le rapport, via le menu Données > Consultation de rapport.



Une fois la requête inscrite, un clic sur bouton Lire les champs va exécuter la requête et vérifier la présence des champs définis dans la requête, pour les inscrire dans le rapport et nous permettre de les utiliser pour la réalisation du modèle de rapport.

Une fois les champs lus, ils sont listés dans la fenêtre Structure du document de iReport :



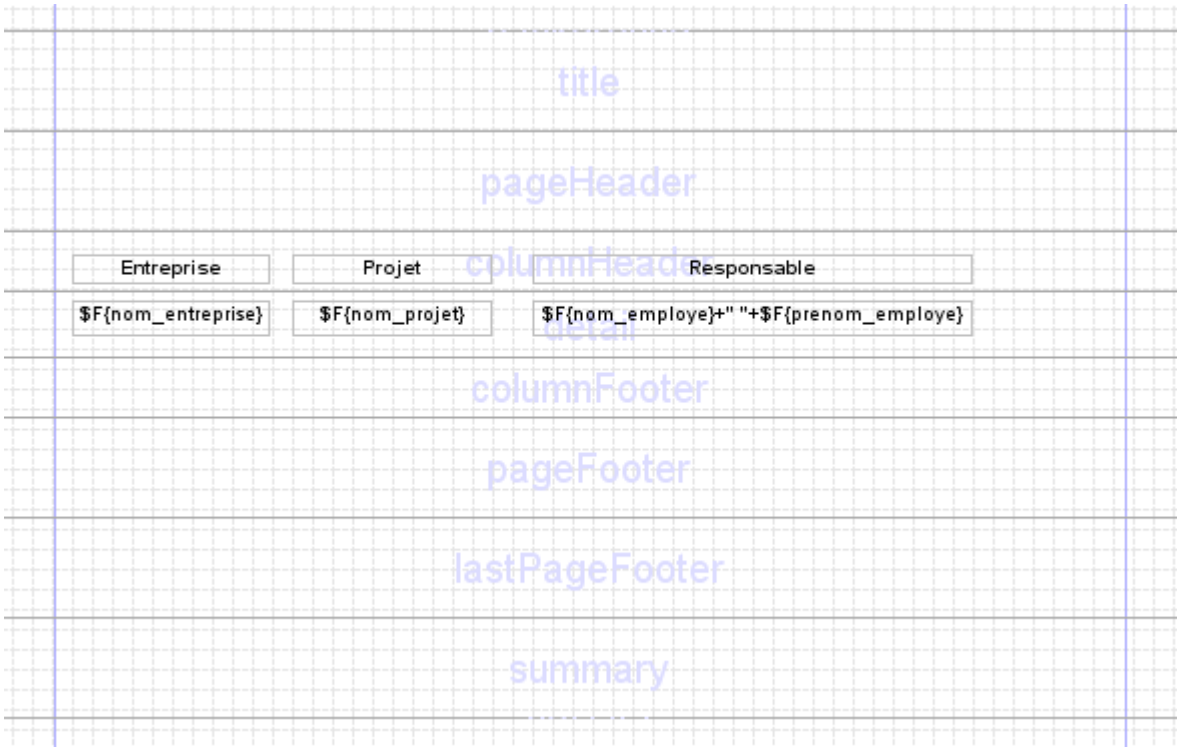
L'ajout de cette requête et de ces champs dans le rapport ajoute les éléments suivants dans le code XML de ce dernier :

```
<queryString>
<![CDATA[SELECT
        p.nom_projet,
        e.nom_entreprise,
        em.nom_employe,
        em.prenom_employe
FROM
        projet p,
        entreprise e,
        employe em
WHERE
        p.id_entreprise = e.id_entreprise AND
        p.id_responsable = em.id_employe]]>
</queryString>
<field name="nom_projet" class="java.lang.String"/>
<field name="nom_entreprise" class="java.lang.String"/>
<field name="nom_employe" class="java.lang.String"/>
<field name="prenom_employe" class="java.lang.String"/>
```

La requête est inscrite dans un tag `queryString`, puis chacun des champs dans un tag `field`, précisant leur type.

4.1.3 MISE EN PAGE DU MODELE

Il ne reste plus maintenant qu'à faire la mis en page de notre modèle.



Une mise en page sous forme de tableau a été réalisée pour présenter les données.

Trois textes statiques ont été ajoutés à la partie columHeader, représentant les entêtes de colonnes du tableau.

Puis dans la partie detail, les champs ajoutés au rapport lors de l'étape précédente ont été ajoutés par glisser/déposer depuis la fenêtre Structure du document.

Attardons-nous sur la syntaxe de l'appel à ces champs :

```
$F{nom_du_champ}
```

Cette syntaxe est une expression, au sens iReport. Les expressions sont proches du langage Java, et permettent, entre autre dans ce cas, un affichage plus sophistiqué des données, comme ici en concaténant la valeur de deux champs.

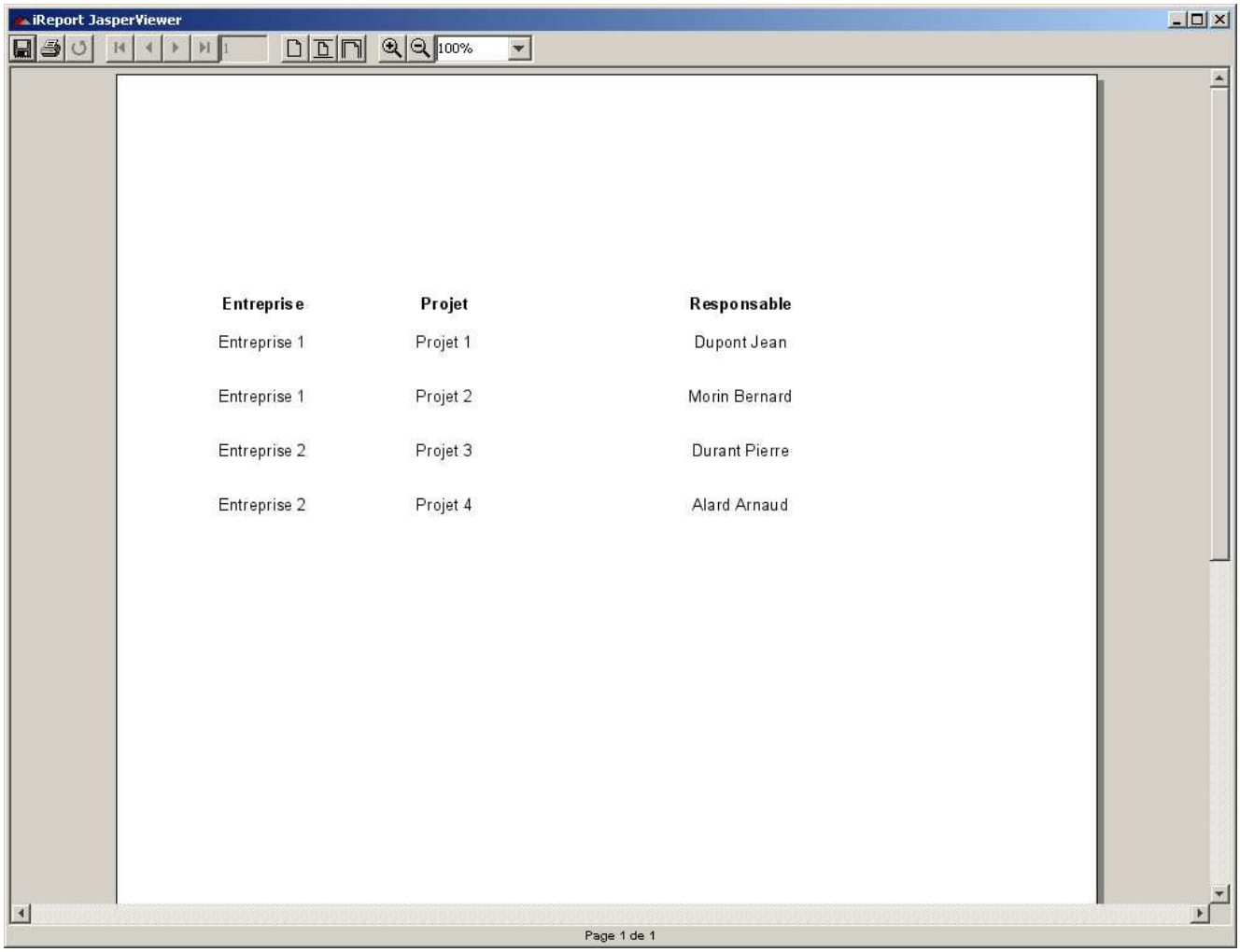
Le code source XML du rapport est modifié de la manière suivante :

```
<columnHeader>
	<band height="30"  isSplitAllowed="true" >
		<staticText>
			<reportElement
				x="9"
				y="12"
				width="99"
				height="15"
				key="staticText-1"/>
			<box></box>
			<textElement textAlignment="Center">
				<font pdfFontName="Helvetica-Bold" isBold="true"/>
```

```
                        </textElement>
                <text><![CDATA[Entreprise]]></text>
                </staticText>
                <staticText>
                        <reportElement
                                x="119"
                                y="12"
                                width="100"
                                height="15"
                                key="staticText-2"/>
                        <box></box>
                        <textElement textAlignment="Center">
                                <font pdfFontName="Helvetica-Bold" isBold="true"/>
                        </textElement>
                <text><![CDATA[Projet]]></text>
                </staticText>
                <staticText>
                        <reportElement
                                x="239"
                                y="12"
                                width="220"
                                height="15"
                                key="staticText-3"/>
                        <box></box>
                        <textElement textAlignment="Center">
                                <font pdfFontName="Helvetica-Bold" isBold="true"/>
                        </textElement>
                <text><![CDATA[Responsable]]></text>
                </staticText>
        </band>
</columnHeader>
<detail>
        <band height="33"  isSplitAllowed="true" >
                <textField isStretchWithOverflow="false" isBlankWhenNull="false"
evaluationTime="Now" hyperlinkType="None"  hyperlinkTarget="Self" >
                        <reportElement
                                x="9"
                                y="5"
                                width="99"
                                height="18"
                                key="textField"/>
                        <box></box>
                        <textElement textAlignment="Center">
                                <font/>
                        </textElement>
                        <textFieldExpression class="java.lang.String">
                                <![CDATA[$F{nom_entreprise}]]>
                        </textFieldExpression>
                </textField>
                <textField isStretchWithOverflow="false" isBlankWhenNull="false"
evaluationTime="Now" hyperlinkType="None"  hyperlinkTarget="Self" >
```

```
                        <reportElement
                                x="119"
                                y="5"
                                width="100"
                                height="18"
                                key="textField"/>
                        <box></box>
                        <textElement textAlignment="Center">
                                <font/>
                        </textElement>
                        <textFieldExpression class="java.lang.String">
                                <![CDATA[$F{nom_projet}]]>
                        </textFieldExpression>
                </textField>
                <textField isStretchWithOverflow="false" isBlankWhenNull="false"
evaluationTime="Now" hyperlinkType="None"  hyperlinkTarget="Self" >
                        <reportElement
                                x="239"
                                y="5"
                                width="220"
                                height="18"
                                key="textField"/>
                        <box></box>
                        <textElement textAlignment="Center">
                                <font/>
                        </textElement>
                        <textFieldExpression class="java.lang.String">
                                <![CDATA[$F{nom_employe}+" "+$F{prenom_employe}]]>
                        </textFieldExpression>
                </textField>
        </band>
</detail>
```

Observons à quoi ressemble notre rapport maintenant, en utilisant la fonctionnalité d'exportation de iReport :



Nous pouvons observer que le contenu de la partie detail est répété pour chaque enregistrement retourné par la requête de sélection incluse dans le rapport.

4.1.4 PARAMETRAGE DE LA REQUETE

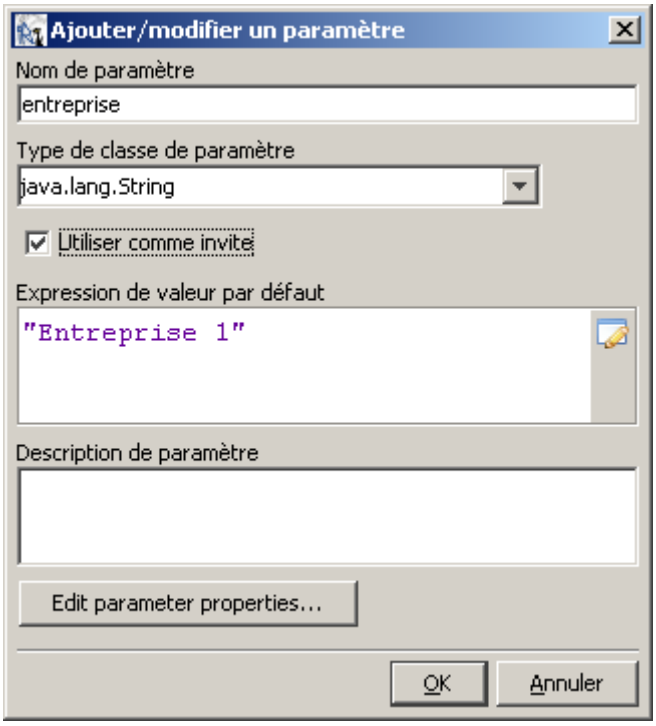
L'inscription de la requête dans le rapport est le moyen le plus rapide de générer un rapport associé à une base de données, mais ce n'est pas solution très flexible. Quand la requête a besoin d'être modifié, il est nécessaire de modifier la source du modèle de rapport au format jrxml.

JasperReports offre la possibilité de paramétrer notre requête et de tirer partie des paramètres de rapport pour personnaliser notre requête à la génération.

Imaginons que l'on souhaite afficher la liste des projets pour une certaine entreprise.

Commençons par ajouter un paramètre à notre rapport, représentant l'entreprise pour laquelle nous voulons afficher les projets.

Dans iReport, on passe par le menu Afficher > Paramètres pour afficher la liste des paramètres du rapport, et on clique sur le bouton Nouveau pour en ajouter un.

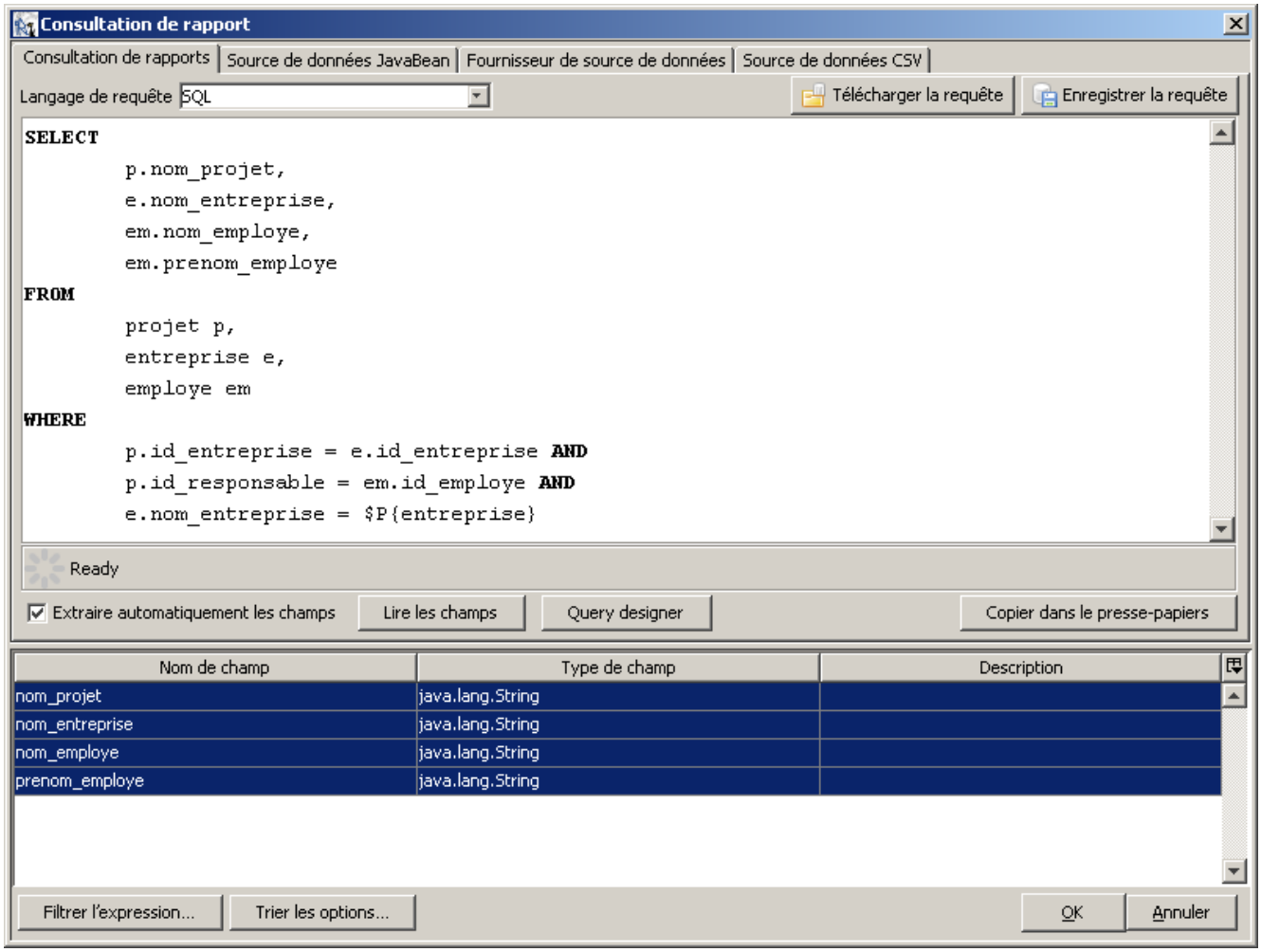


L'assistant nous permet d'indiquer le nom du paramètre, son type, sa valeur par défaut, et enfin, par l'intermédiaire d'une case à cocher, si l'on souhaite qu'à l'exécution dans iReport, une fenêtre de dialogue nous demande la valeur que l'on souhaite donner à ce paramètre (dans le cas contraire, c'est la valeur par défaut qui sera utilisée).

On retrouve ce paramètre dans le code source du modèle de rapport :

```
<parameter name="entreprise" isForPrompting="true" class="java.lang.String">
	<defaultValueExpression>
		<![CDATA["Entreprise 1"]]>
	</defaultValueExpression>
</parameter>
```

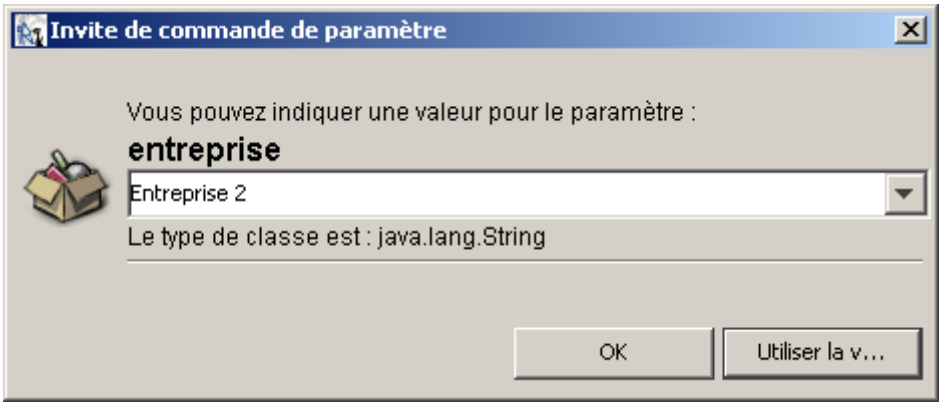
Nous allons ensuite pouvoir ensuite utiliser ce paramètre dans la requête SQL de sélection des données :

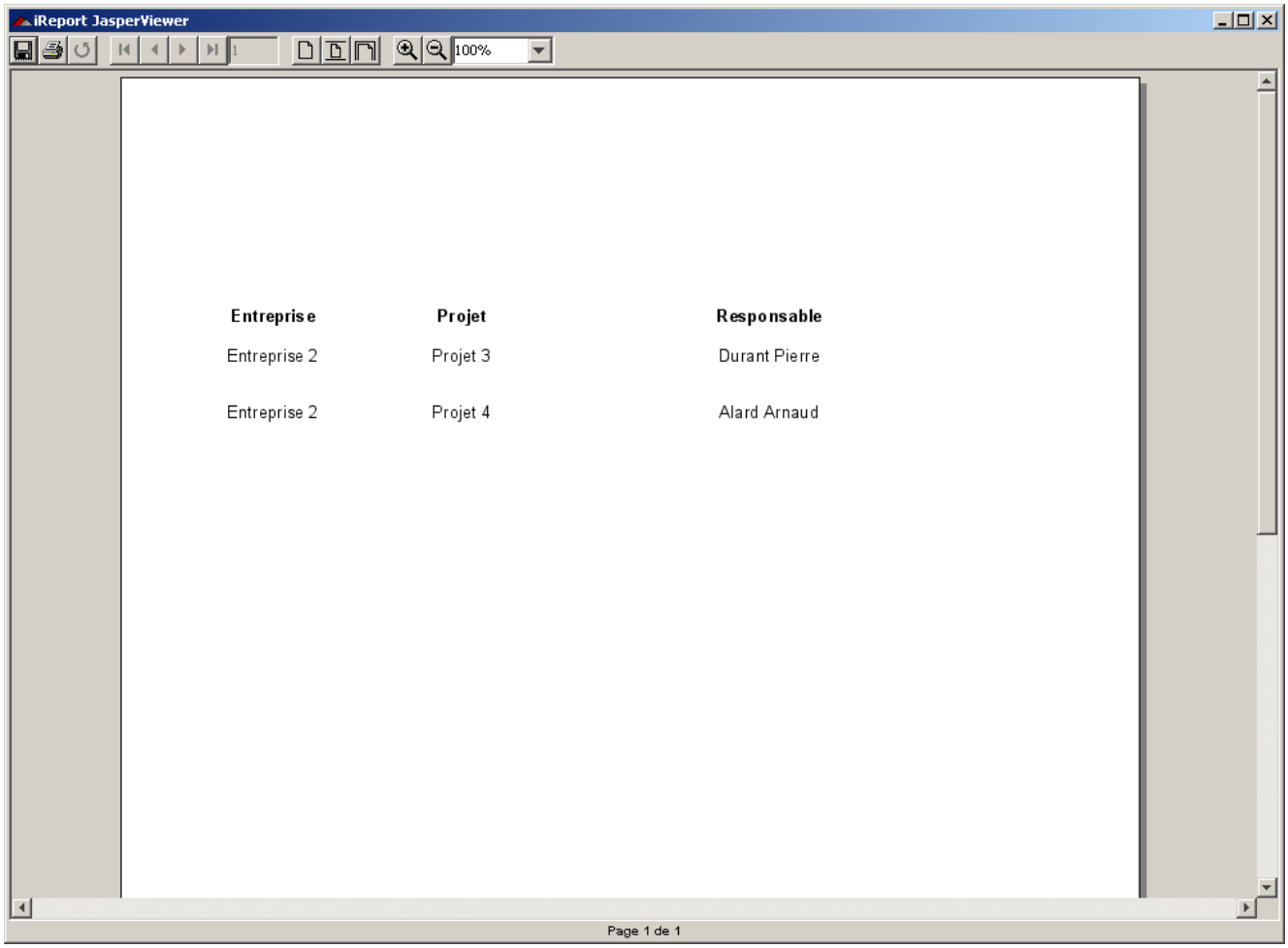


On remarque la syntaxe de l'appel à un paramètre :

```
$P{nom_du_paramètre}
```

Utilisons la fonctionnalité d'export de iReport pour voir notre rapport. A l'exécution, si nous l'avons demandé lors de la création du paramètre, une fenêtre de dialogue demandera la valeur que l'on souhaite donner au paramètre entreprise :





Dans le cadre de la génération d'un rapport par une application Java, la valeur des paramètres d'un rapport est renseignée sous la forme d'un objet de type `Map`, dans lequel pour chaque enregistrement le nom du paramètre est la clé. Cette Map est ensuite passée en paramètre à la méthode de remplissage du rapport, en même temps que la connexion JDBC à la base.

4.2 RAPPORT ASSOCIE A UNE SOURCE DE DONNEES

Nous avons comment un rapport peut utiliser une connexion JDBC pour récupérer les données à afficher sur un rapport dynamique. Un autre moyen pour fournir des données à un rapport est d'utiliser une datasource (source de données). Une datasource est une classe implémentant l'interface `net.sf.jasperreports.engine.JRDataSource`.

JasperReports fournit plusieurs implémentations de cette interface, permettant d'utiliser comme source de données divers éléments :

- des resultSets JDBC
- Des objets de type Map
- Des beans Java
- Des fichiers XML

- …

Une implémentation particulière également proposée est la datasource vide (JREmptyDataSource). En effet, il n'est pas possible de créer un rapport sans fournir une connexion ou une datasource. La datasource vide sera ainsi un moyen de fournir une source de données à un rapport entièrement statique, ou dont les éléments dynamiques ne serait fournis au rapport que sous forme de paramètres.

Nous allons décrire dans ce chapitre l'utilisation comme source de données d'objets Java, et d'un fichier XML.

4.2.1 RAPPORT ASSOCIE A DES OBJETS JAVA

Les objets Java utilisés pour fournir des données au rapport seront de simples JavaBeans. Il devra particulièrement proposer des accesseurs et modificateurs (getters et des setters) pour toutes ces propriétés. Voila par exemple le code source d'un JavaBean représentant une entreprise, sur le modèle du schéma de base de données du paragraphe 4.1 :

```
public class Entreprise {

	/**
	 * Identifiant de l'entreprise
	 */
	private Integer idEntreprise = null;

	/**
	 * Nom de l'entreprise
	 */
	private String nomEntreprise = null;

	/**
	 * Constructeur
	 */
	public Entreprise() {

	}

	/**
	 * Constructeur
	 * @param idEntreprise l'identifiant de l'entreprise
	 * @param nomEntreprise le nom de l'entreprise
	 */
	public Entreprise(Integer idEntreprise, String nomEntreprise) {
		this.idEntreprise = idEntreprise;
		this.nomEntreprise = nomEntreprise;
	}

	/**
	 * Setter de idEntreprise
	 * @param idEntreprise l'identifiant de l'entreprise
```

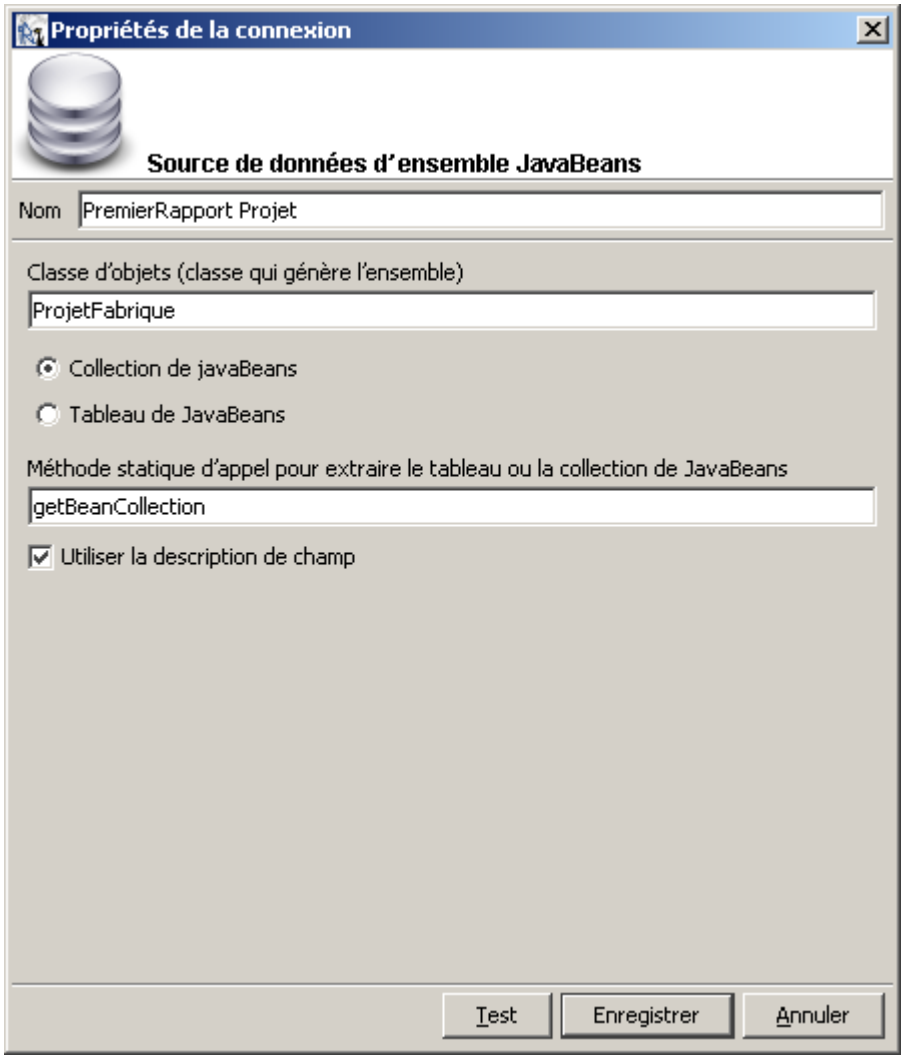
```
         */
        public void setIdEntreprise(Integer idEntreprise) {
                this.idEntreprise = idEntreprise;
        }

        /**
         * Getter de idEntreprise
         * @return l'identifiant de l'entreprise
         */
        public Integer getIdEntreprise() {
                return idEntreprise;
        }

        /**
         * Setter de nomEntreprise
         * @param nomEntreprise le nom de l'entreprise
         */
        public void setNomEntreprise(String nomEntreprise) {
                this.nomEntreprise = nomEntreprise;
        }

        /**
         * Getter de nomEntreprise
         * @return le nom de l'entreprise
         */
        public String getNomEntreprise() {
                return nomEntreprise;
        }

}
```

4.2.1.1 CONFIGURATION DE LA SOURCE DE DONNÉES

Nous allons configurer iReport afin qu'il puisse utiliser nos JavaBeans comme datasource, en passant par le menu Données > Connexions/Sources de données, puis le bouton Nouveau, et enfin en choisissant Source de données d'ensemble JavaBeans dans la liste.

Un assistant nous permet de saisir le nom de la fabrique et de la méthode permettant à iReport de récupérer l'ensemble des instances des JavaBeans. Cet ensemble peut être fournit sous deux formes, correspondants à deux implémentations différentes (fournis par JasperReports) d'une source de données composées de JavaBeans :

- Sous le forme d'un tableau, pour une datasource de type `JRBeanArrayDataSource`
- Sous la forme d'un objet de type `Collection`, pour une datasource de type `JRBeanCollectionDataSource`

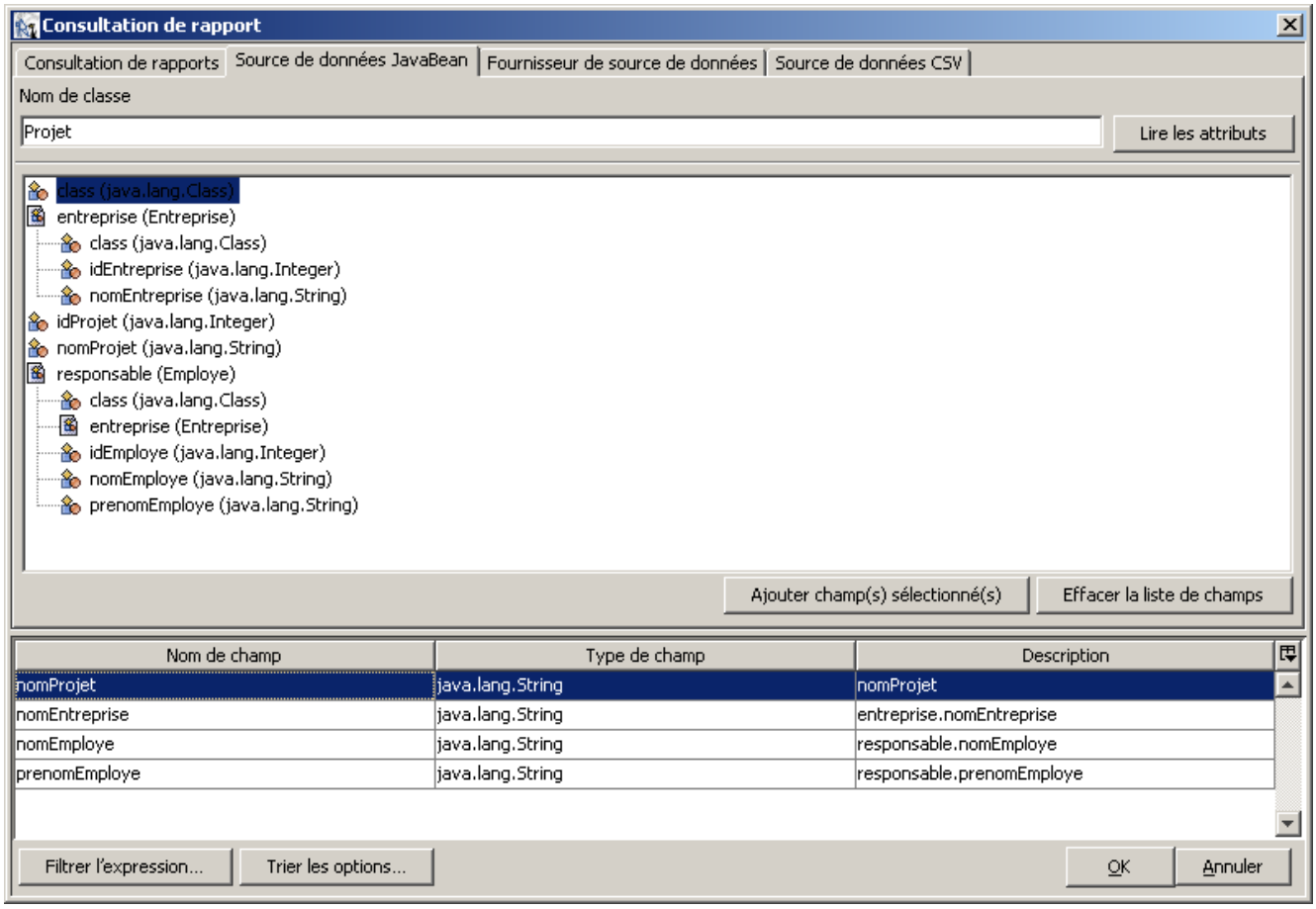


iReport se chargera de créer l'instance de la datasource correspondante au type de retour de la fabrique, à partir de ce retour.

Dans le cadre de la génération d'un rapport par une application Java, cette datasource sera créée dans le code puis passée en paramètre à la méthode de remplissage du rapport.

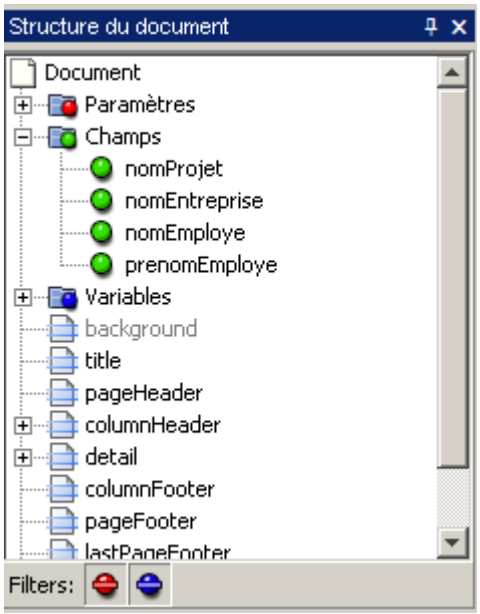
4.2.1.2 INSCRIPTION DES PROPRIÉTÉS DES OBJETS

De manière similaire à l'inscription de la requête SQL dans le chapitre précédent, le menu Données > Consultation de rapport va nous permettre d'inscrire le type d'objets que la datasource va fournir au rapport et d'ajouter facilement les propriétés de cette classe dans les champs du rapport :



On remarquera que JasperReports permet d'utiliser un objet comme propriété d'un JavaBeans, et est capable d'aller chercher les propriétés de cet objet pour les ajouter comme champ du rapport.

Les propriétés sélectionnées sont maintenant listés dans la fenêtre Structure du document de iReport :

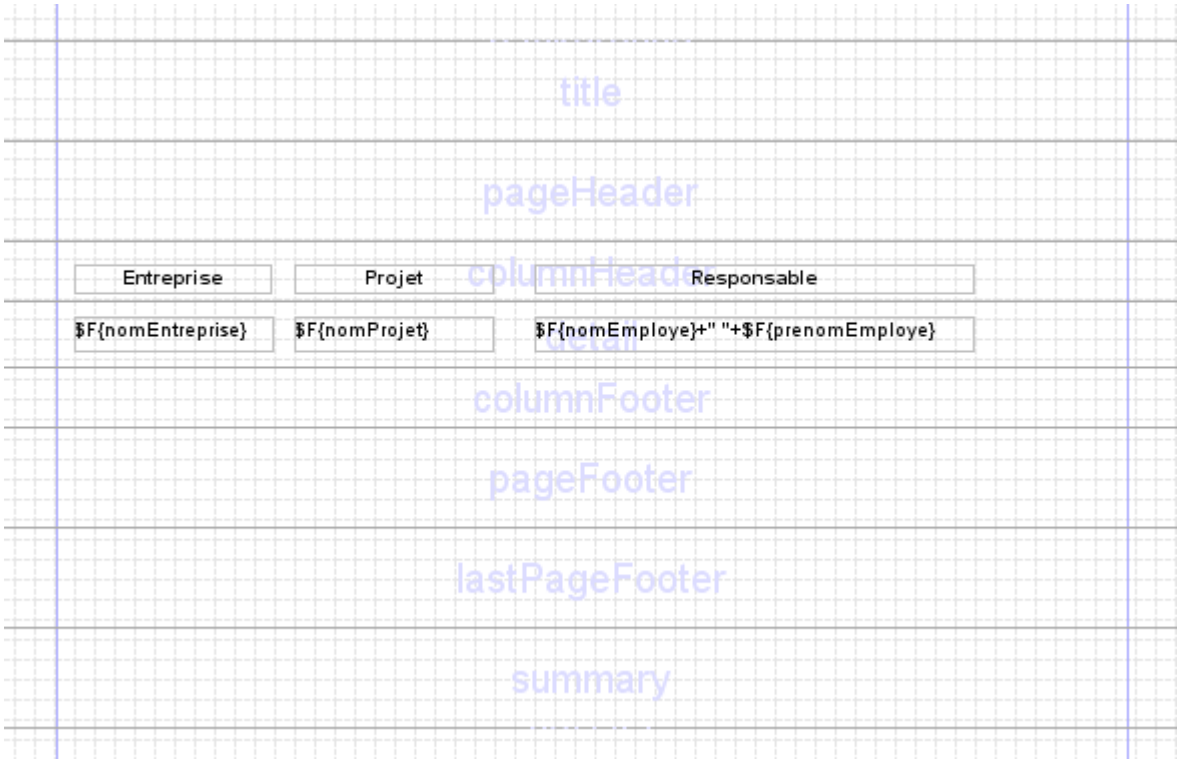


On peut retrouver ces champs dans le code source du modèle de rapport :

```
<field name="nomProjet" class="java.lang.String">
	<fieldDescription><![CDATA[nomProjet]]></fieldDescription>
</field>
<field name="nomEntreprise" class="java.lang.String">
	<fieldDescription><![CDATA[entreprise.nomEntreprise]]></fieldDescription>
</field>
<field name="nomEmploye" class="java.lang.String">
	<fieldDescription><![CDATA[responsable.nomEmploye]]></fieldDescription>
</field>
<field name="prenomEmploye" class="java.lang.String">
	<fieldDescription><![CDATA[responsable.prenomEmploye]]></fieldDescription>
</field>
```

4.2.1.3 MISE EN PAGE DU MODÈLE

Par rapport à notre exemple précédent, les champs ne portent plus les mêmes noms, le modèle de rapport va ainsi être modifié en conséquence :



Le résultat obtenu sera le même que dans le chapitre précédent.

4.2.2 RAPPORT ASSOCIE A UN FICHIER XML

Un document XML peut également servir de source de données. Dans ce cas, c'est une expression XPath qui servira à extraire les données à destination du rapport. Le document XML suivant sera utilisé comme source de données pour la suite de ce chapitre (il s'agit d'une simple sérialisation du modèle objet utilisé au chapitre précédent) :

```
<?xml version="1.0" encoding="UTF-8"?>
<projetData>
	<projet>
		<idProjet>1</idProjet>
		<nomProjet>Projet 1</nomProjet>
		<entreprise>
			<idEntreprise>1</idEntreprise>
			<nomEntreprise>Entreprise 1</nomEntreprise>
		</entreprise>
		<responsable>
			<idEmploye>1</idEmploye>
			<entreprise>
				<idEntreprise>1</idEntreprise>
				<nomEntreprise>Entreprise 1</nomEntreprise>
			</entreprise>
			<nomEmploye>Dupont</nomEmploye>
			<prenomEmploye>Jean</prenomEmploye>
		</responsable>
	</projet>
	<projet>
		<idProjet>2</idProjet>
		<nomProjet>Projet 2</nomProjet>
		<entreprise>
			<idEntreprise>1</idEntreprise>
			<nomEntreprise>Entreprise 1</nomEntreprise>
		</entreprise>
		<responsable>
			<idEmploye>2</idEmploye>
			<entreprise>
				<idEntreprise>1</idEntreprise>
				<nomEntreprise>Entreprise 1</nomEntreprise>
			</entreprise>
			<nomEmploye>Morin</nomEmploye>
			<prenomEmploye>Bernard</prenomEmploye>
		</responsable>
	</projet>
	<projet>
		<idProjet>3</idProjet>
		<nomProjet>Projet 3</nomProjet>
		<entreprise>
			<idEntreprise>2</idEntreprise>
			<nomEntreprise>Entreprise 2</nomEntreprise>
		</entreprise>
		<responsable>
```

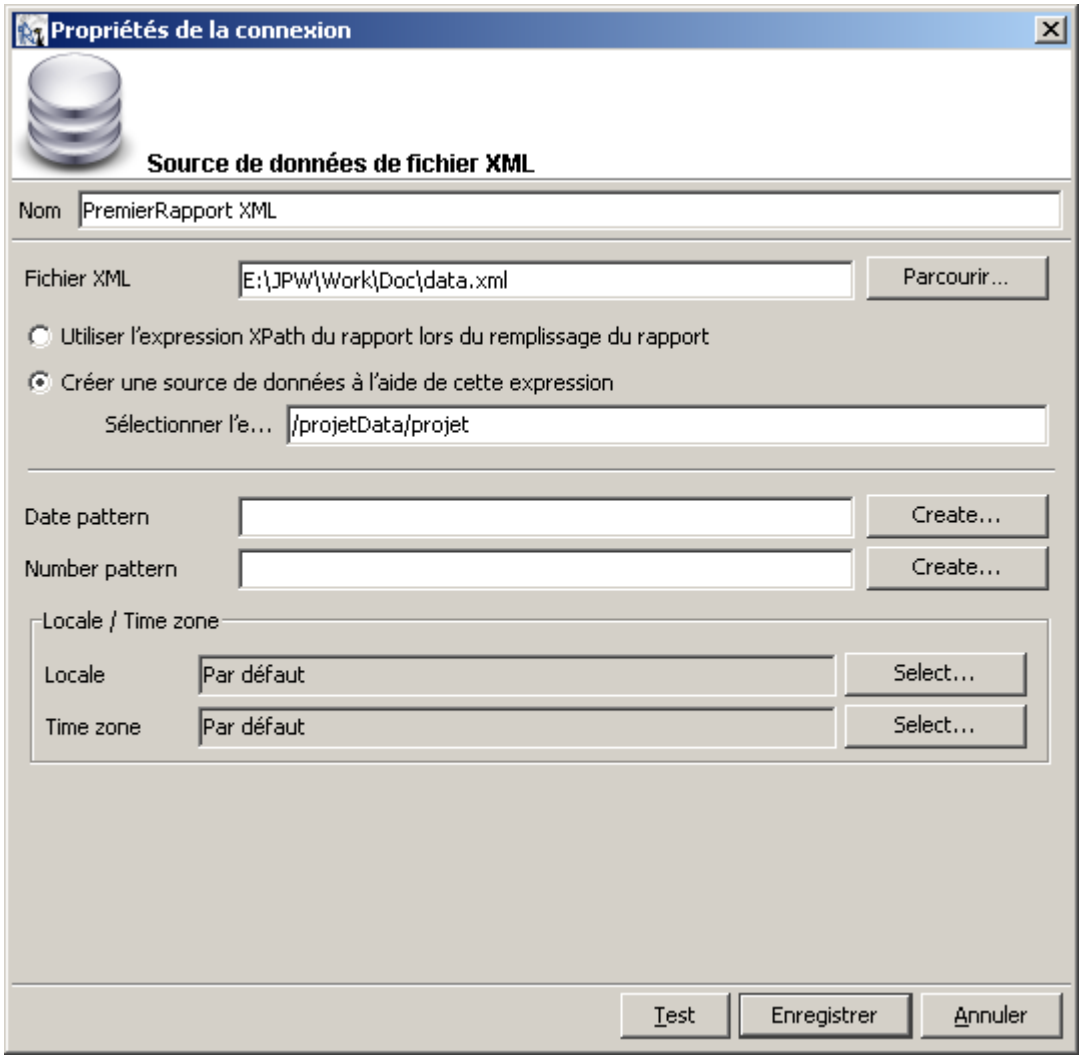
```
                        <idEmploye>3</idEmploye>
                        <entreprise>
                                <idEntreprise>2</idEntreprise>
                                <nomEntreprise>Entreprise 2</nomEntreprise>
                        </entreprise>
                        <nomEmploye>Durant</nomEmploye>
                        <prenomEmploye>Pierre</prenomEmploye>
                </responsable>
        </projet>
        <projet>
                <idProjet>4</idProjet>
                <nomProjet>Projet 4</nomProjet>
                <entreprise>
                        <idEntreprise>2</idEntreprise>
                        <nomEntreprise>Entreprise 2</nomEntreprise>
                </entreprise>
                <responsable>
                        <idEmploye>4</idEmploye>
                        <entreprise>
                                <idEntreprise>2</idEntreprise>
                                <nomEntreprise>Entreprise 2</nomEntreprise>
                        </entreprise>
                        <nomEmploye>Alart</nomEmploye>
                        <prenomEmploye>Arnaud</prenomEmploye>
                </responsable>
        </projet>
</projetData>
```

4.2.2.1 CONFIGURATION DE LA SOURCE DE DONNÉES

Nous allons configurer iReport afin qu'il puisse utiliser ce fichier XML comme datasource, en passant par le menu Données > Connexions/Sources de données, puis le bouton Nouveau, et enfin en choisissant Source de données de fichier XML dans la liste.

Un assistant nous permet de saisir le chemin vers le fichier XML, puis de choisir comment sera fournit la requête XPath permettant d'accéder aux données :

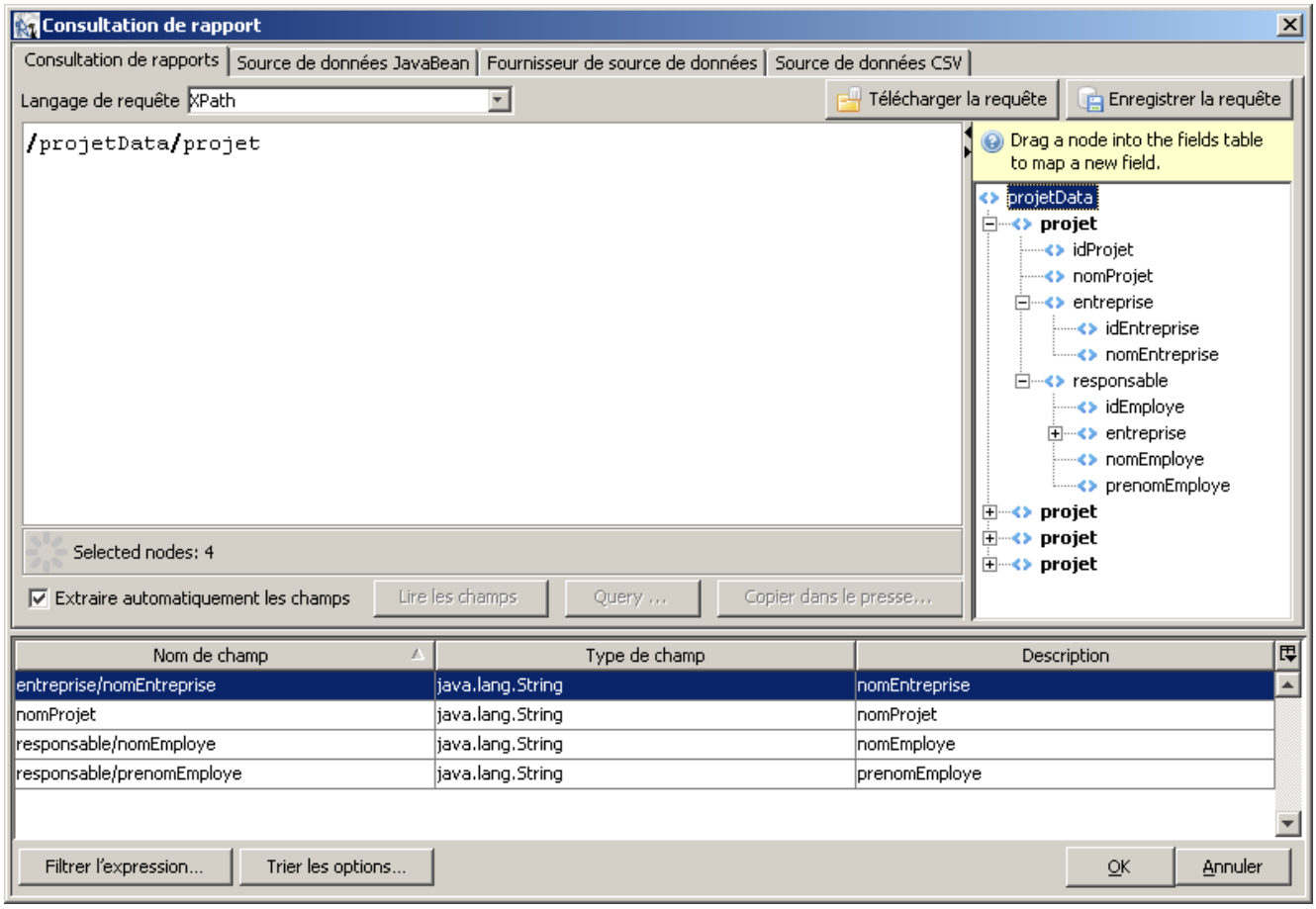
- Soit en passant la requête au rapport lors du remplissage de celui-ci. Cette solution permettra, dans le cadre de la génération d'un rapport par une application Java, de faire passer la requête au rapport par le code java
- Soit en inscrivant la requête dans le rapport, à la manière de l'inscription de la requête SQL dans le chapitre 4.1.2



Dans le cadre de la génération d'un rapport par une application Java, le chemin vers le fichier XML, ou directement le contenu du fichier, sera passé en paramètre à la méthode de remplissage du rapport, en compagnie ou non de la requête XPath, en fonction du choix décrit ci-dessus.

4.2.2.2 INSCRIPTION DES NOEUDS

De manière similaire à l'inscription de la requête SQL ou des propriétés des objets Java dans les chapitres précédents, le menu Données > Consultation de rapport va nous permettre d'inscrire les nœuds du fichier XML que la datasource va utiliser comme champs du rapport :



L'ajout de ces champs dans la fenêtre Structure du document de iReport est identique au chapitre précédent, et le résultat dans le code source sera le suivant :

```
<queryString language="xPath"><![CDATA[/projetData/projet]]></queryString>

<field name="nomProjet" class="java.lang.String">
	<fieldDescription><![CDATA[nomProjet]]></fieldDescription>
</field>
<field name="nomEntreprise" class="java.lang.String">
	<fieldDescription><![CDATA[entreprise/nomEntreprise]]></fieldDescription>
</field>
<field name="nomEmploye" class="java.lang.String">
	<fieldDescription><![CDATA[responsable/nomEmploye]]></fieldDescription>
</field>
<field name="prenomEmploye" class="java.lang.String">
	<fieldDescription><![CDATA[responsable/prenomEmploye]]></fieldDescription>
</field>
```

On retrouve le tag queryString, accompagné de l'attribut langage, qui inscrit la requête Xpath dans le rapport. Ce tag ne sera présent que si l'ajout de le requête dans le rapport y est choisit dans l'assistant.

Les tags du fichier XML ayant le même nom que les propriétés des objets Java du chapitre précédent, la mise en page du modèle de rapport n'a pas à être modifiée pour obtenir le même résultat que dans les chapitres précédents.

5 PARAMETRAGE ET MISE EN PAGE D'UN RAPPORT – FONCTIONNALITES AVANCEES

5.1 COLONNES

JasperReports permet de générer des rapports possédant plusieurs colonnes.

Dans ce contexte, le contenu des sections `columnHeader` et `columnFooter` sont répétés pour chacune des colonnes, et le contenu de la section `detail` devra être considéré comme une cellule de ces colonnes.

En gardant notre modèle précédent, affichons la liste de tous les employés du modèle, dans un rapport à 3 colonnes.

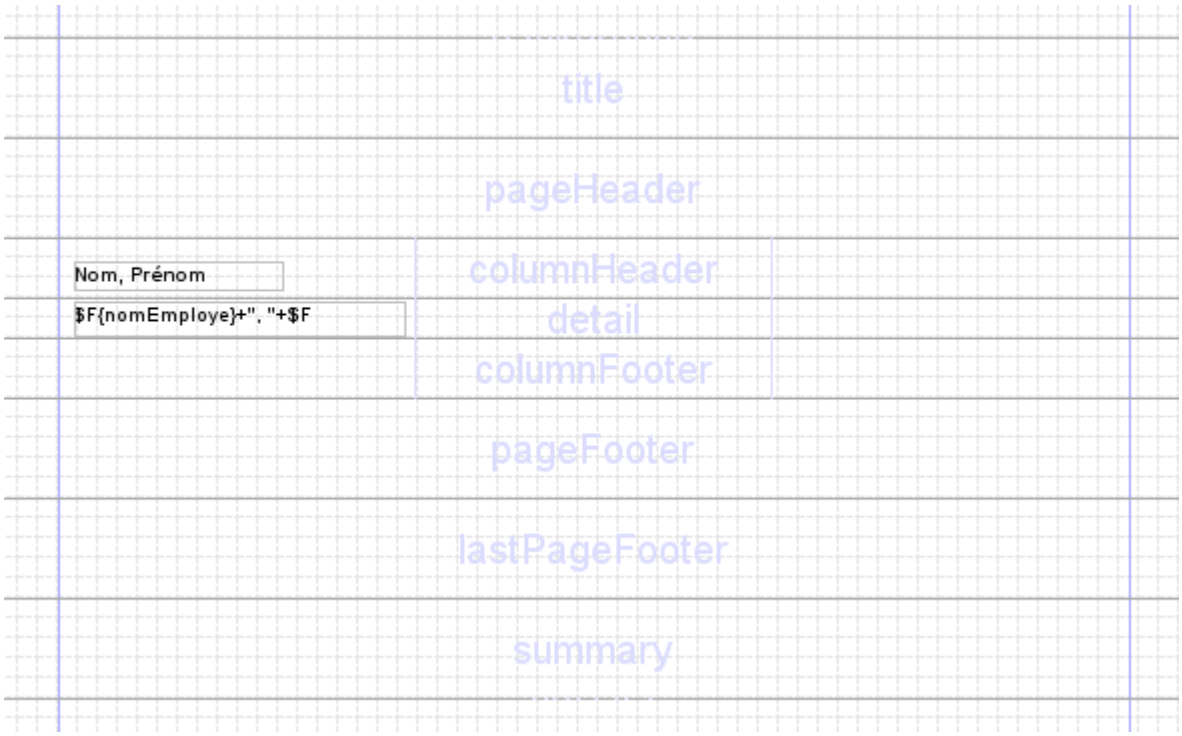
Dans la fenêtre des propriétés du document, on indique le nombre de colonne dont on veut composer le document. La largeur de colonne est mis automatiquement à jour pour obtenir 3 colonnes de même largeur.

Colonnes du rapport	
Colonnes	1
Largeur	535
Espacement	0

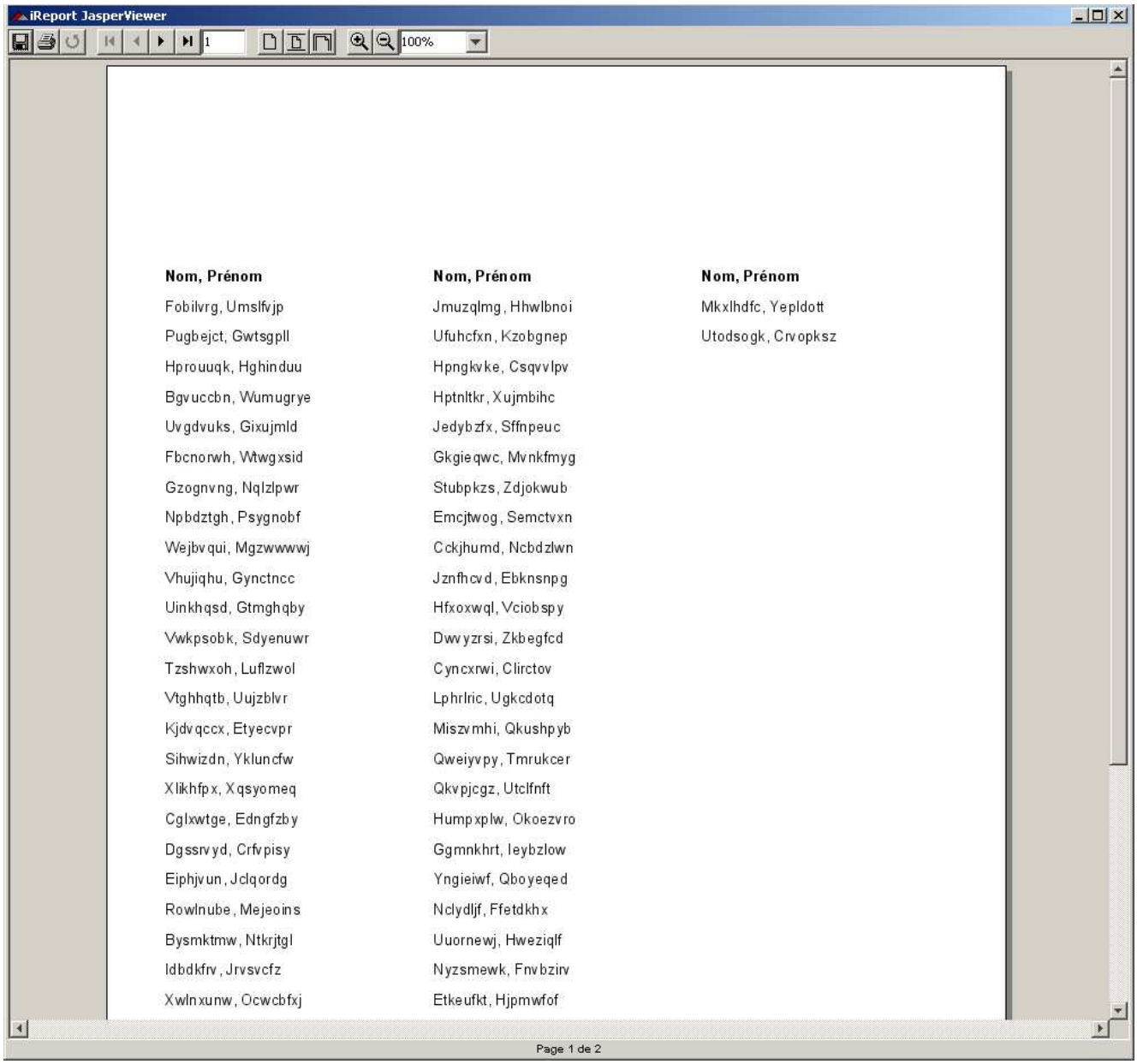
→

Colonnes du rapport	
Colonnes	3
Largeur	178
Espacement	0

Le contenu d'une colonne est ensuite défini :



Pour le résultat suivant :



Dans le code source du rapport, la définition du nombre de colonnes se fait dans la tag principal du document :

```
<jasperReport
        name="PremierRapport"
        columnCount="3"
        printOrder="Vertical"
        orientation="Portrait"
        pageWidth="595"
        pageHeight="842"
        columnWidth="178"
        columnSpacing="0"
```

```
        leftMargin="30"
        rightMargin="30"
        topMargin="20"
        bottomMargin="20"
        whenNoDataType="BlankPage"
        isTitleNewPage="false"
        isSummaryNewPage="false">
```

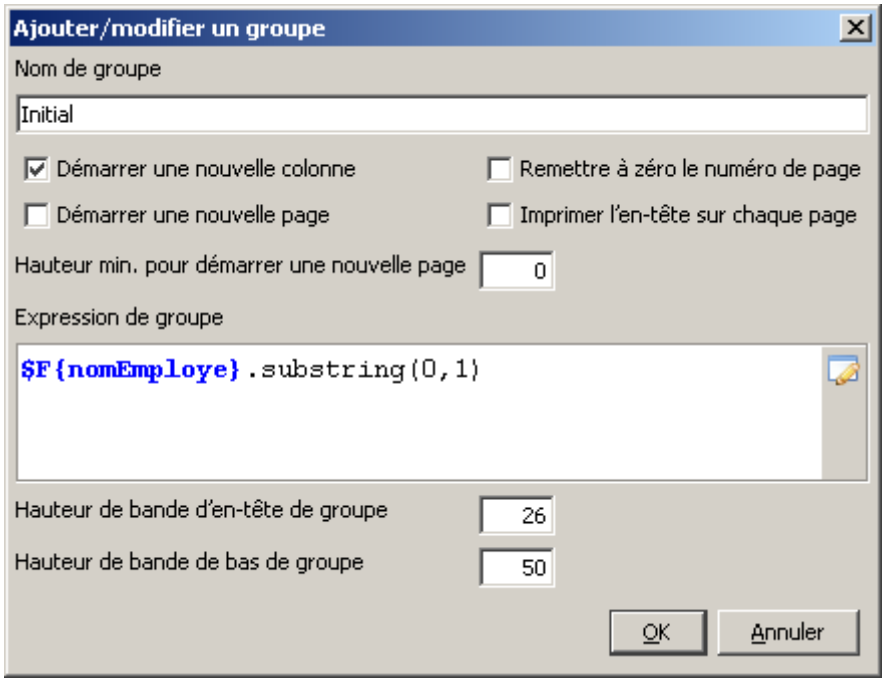
On y remarque l'attribut `printOrder`. Celui permet d'indiquer le sens de remplissage du rapport, vertical comme dans notre exemple, ou horizontal, en lui donnant la valeur « Horizontal ».

5.2 GROUPE DE DONNEES

JasperReports permet d'effectuer des regroupements logiques sur les données de la datasource d'un rapport, afin d'inclure une rupture entre chacun de ces groupes (nouvelle colonne ou nouvelle page par exemple.

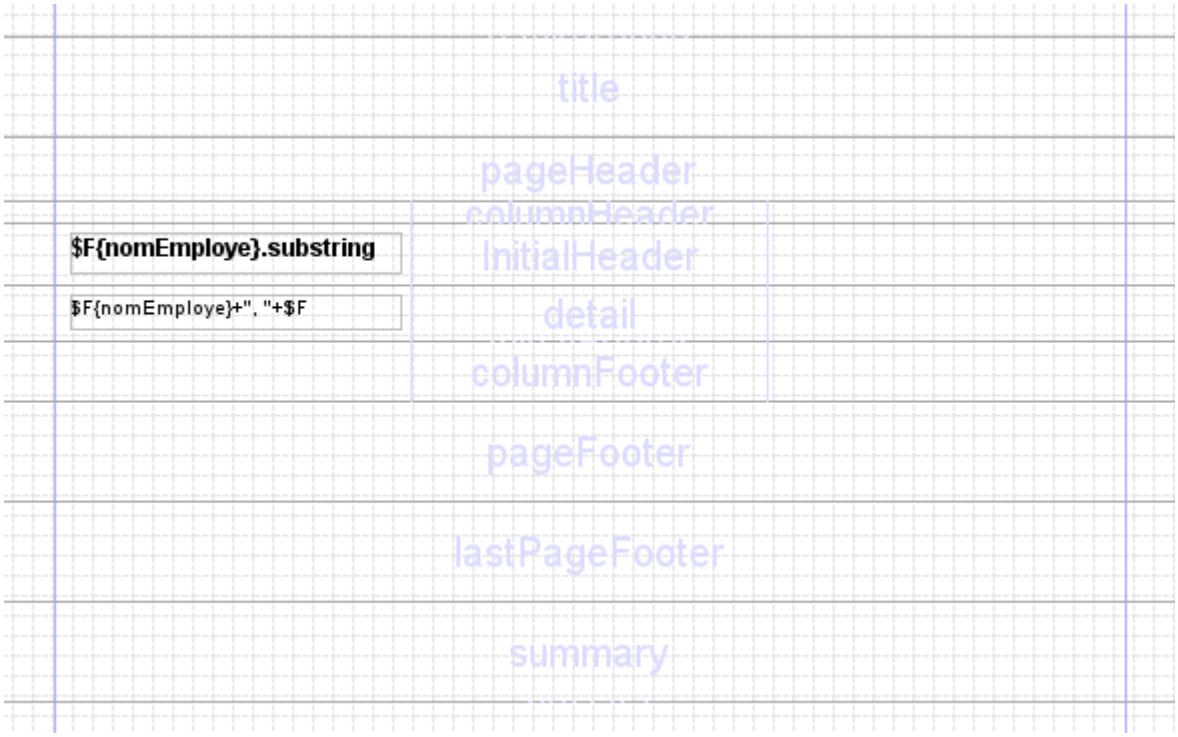
Reprenons comme exemple la liste d'employés du chapitre précédent. Nous souhaitons maintenant afficher ces employés regroupés par la première lettre de leur nom de famille. Chaque lettre sera listée dans une colonne différente.

Définissons un nouveau groupe de données basé sur la première lette du nom de l'employé. Dans iReport, on accède à l'interface de création de groupe par l'icône  , ou par le menu Afficher > Groupe de rapport. On clique sur Nouveau et on arrive sur l'interface de création d'un groupe.



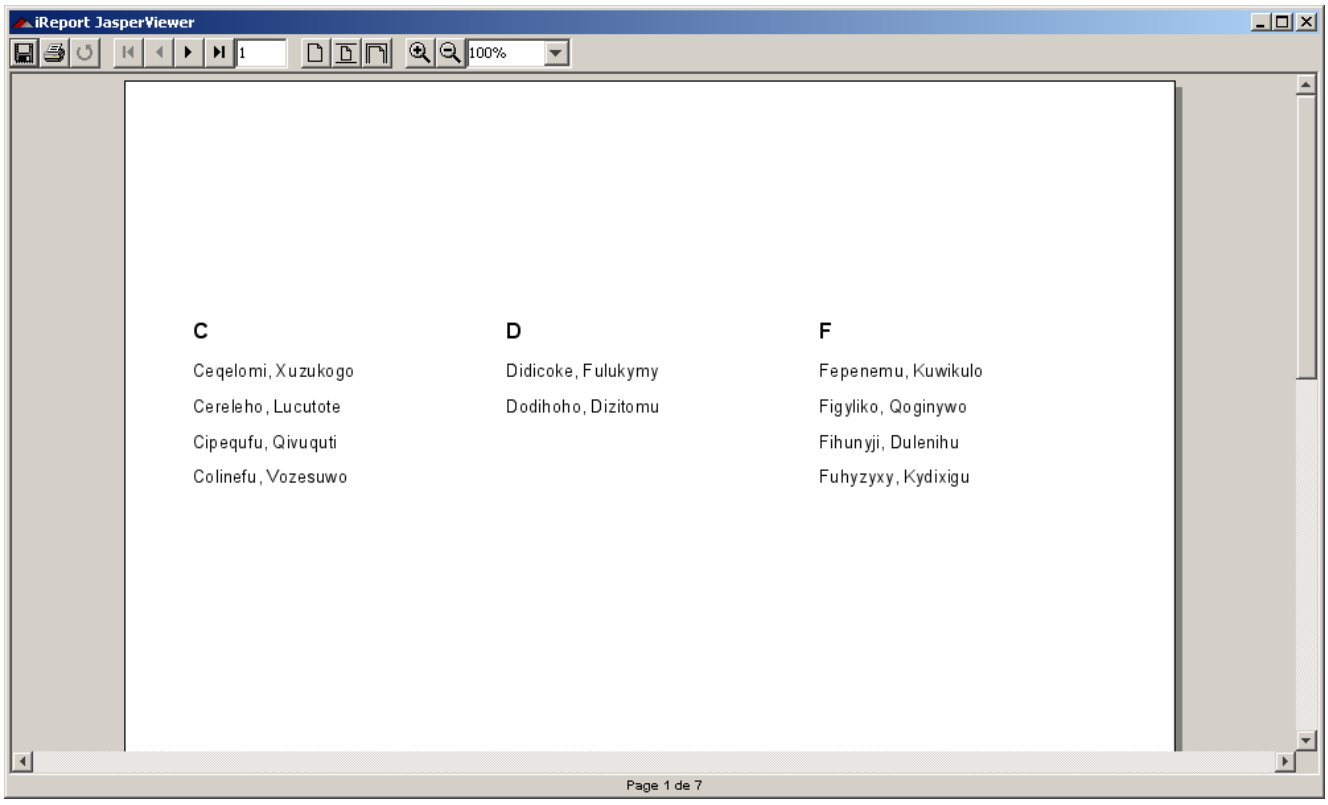
Le regroupement ce fait sur la base d'une expression. Plusieurs options de rupture de mise en page existent, nous choisissons de démarrer une nouvelle colonne.

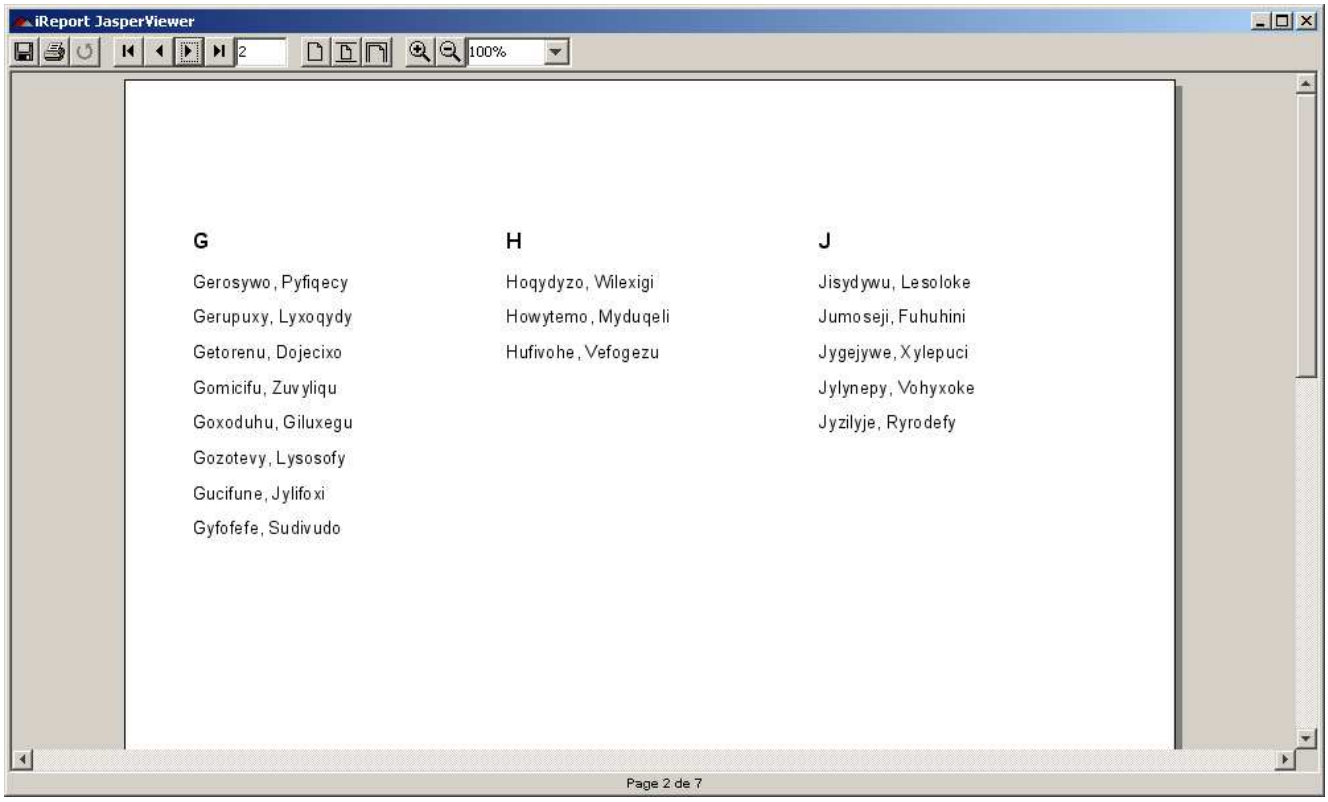
Le modèle de rapport est légèrement modifié pour afficher l'initiale en entête de chaque groupe :



Cet entête est inscrit dans un entête de groupe. Pour chaque groupe, on peut optionnellement définir un entête et un pied de groupe afin d'afficher un texte ou autre au début et à la fin de l'affichage de chaque groupe.

Le résultat obtenu est le suivant :





...

La définition du groupe dans le code source du modèle de rapport à la forme suivante :

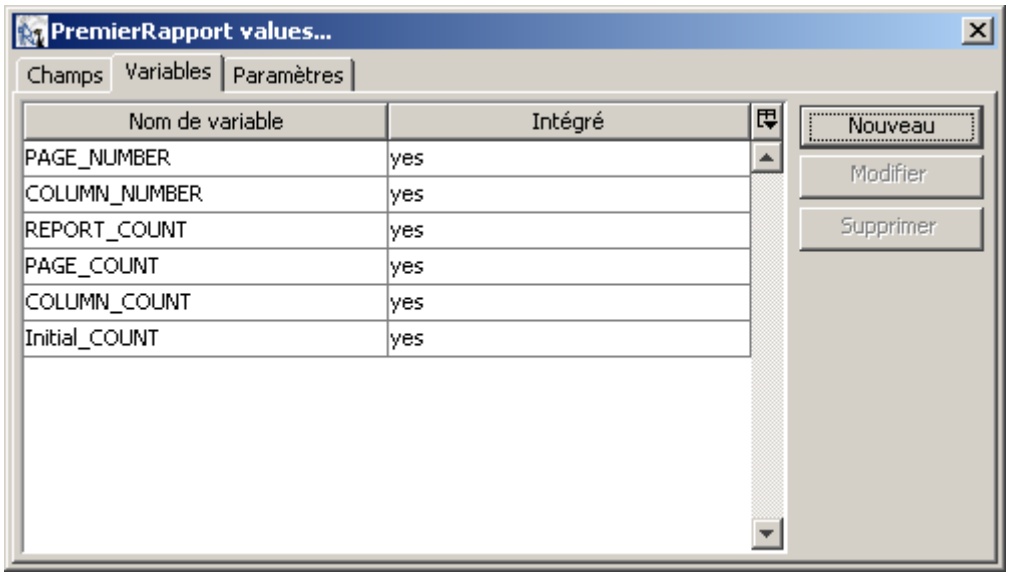
```
<group  name="Initial" isStartNewColumn="true" >
        <groupExpression>
                <![CDATA[$F{nomEmploye}.substring(0,1)]]>
        </groupExpression>
        <groupHeader>
                <band height="31"  isSplitAllowed="true" >
                        <textField isStretchWithOverflow="false"
isBlankWhenNull="false" evaluationTime="Now" hyperlinkType="None"
hyperlinkTarget="Self" >
                                <reportElement
                                        x="8"
                                        y="5"
                                        width="166"
                                        height="21"
                                        key="textField-1"/>
                                <box></box>
                                <textElement>
                                        <font pdfFontName="Helvetica-Bold" size="12"
isBold="true"/>
                                </textElement>
                                <textFieldExpression   class="java.lang.String">
```

```
                                        <![CDATA[$F{nomEmploye}.substring(0,1)]]>
                                </textFieldExpression>
                        </textField>
                </band>
        </groupHeader>
        <groupFooter>
                <band height="0"  isSplitAllowed="true" >
                </band>
        </groupFooter>
</group>
```

5.3 VARIABLES

Afin de factoriser le calcul d'une expression pouvant être réaliser plusieurs fois dans un même rapport, JasperReports offre la possibilité de déclarer des variables au sein d'un rapport.

On accède à l'écran de gestion des variables par l'icône  ou le menu Afficher > Variables.

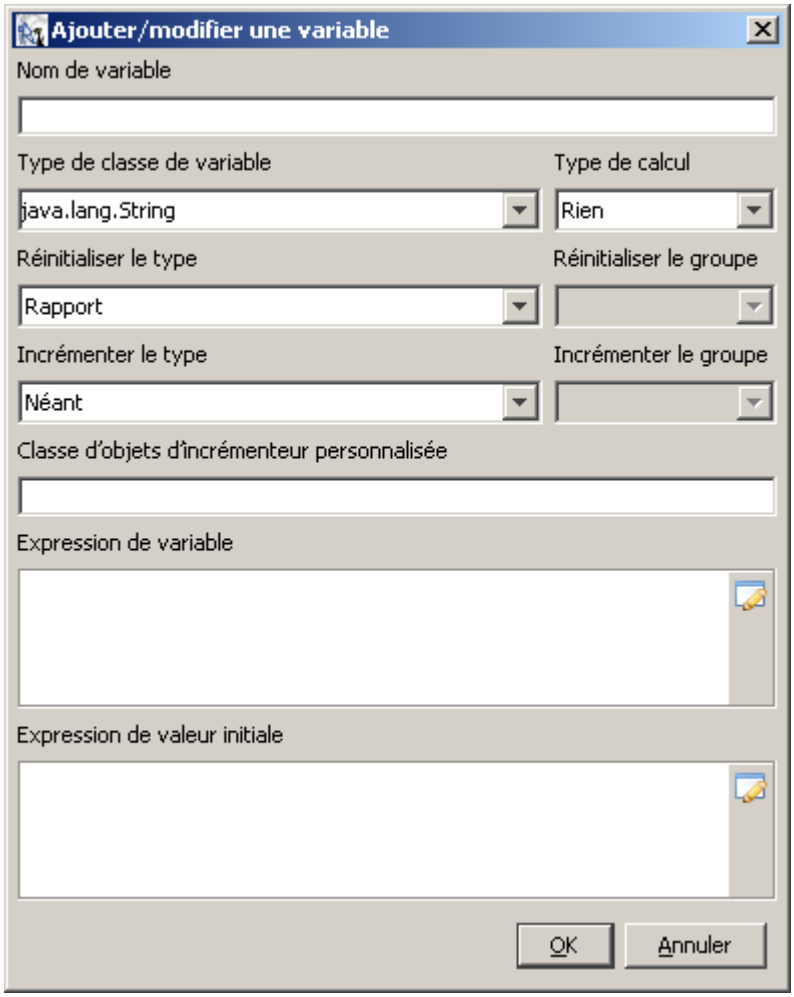


On remarque dès à présent les variables intégrés à chaque rapport :

- PAGE_NUMBER : contient le nombre total de pages du rapport
- COLUMN_NUMBER : contient le nombre courant de colonnes
- REPORT_COUNT : contient le nombre d'enregistrements déjà traités
- PAGE_COUNT : contient le nombre d'enregistrements déjà traités dans la page courante
- COLUMN_COUNT : contient le nombre d'enregistrement déjà traités dans la colonne courante
- <nomGroupe>_COUNT : contient le nombre d'enregistrement déjà traités dans la groupe courant

Ces variables seront uniquement accessible en lecture pour l'utilisateur.

L'utilisateur peut définir ses propres variables en cliquant sur le bouton Nouveau :



Plusieurs attributs permettent un contrôle sur l'évolution de la valeur de la variable, notamment :

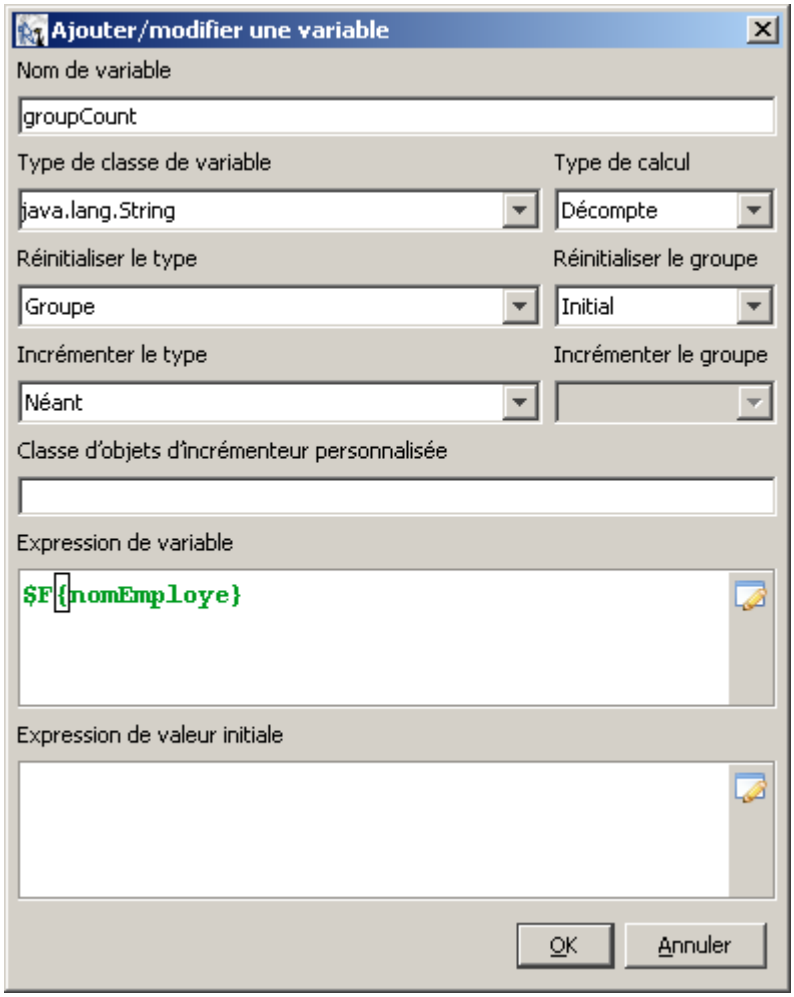
- Type de calcul : indique que type de calcul va être fait sur la variable lors du remplissage du rapport. Les valeurs possibles sont :
  - Rien (pas de calcul)
  - Décompte
  - Décompte distinct
  - Somme
  - Moyenne
  - Plus basse
  - Plus élevé
  - Ecart standard
  - Ecart
  - Système (calcul défini par l'utilisateur, par exemple via une scriptlet)
  - Premier (la première valeur rencontrée)

- Réinitialiser le type : indique à quelle occasion la variable doit être réinitialiser. Les valeurs possibles sont :
    - Néant (pas d'initialisation)
    - Rapport (initialiser une fois au début du rapport)
    - Page (initialiser au début de chaque page)
    - Colonne (initialiser au début de chaque colonne)
    - Groupe (initialiser au début de chaque groupe) (pour ce dernier choix, on indiquera sur quel groupe la variable doit être réinitialisée dans l'attribut Réinitialiser le groupe)
- Incrémenter le type : indique à quelle occasion la variable doit être incrémenter. Les valeurs possibles sont les mêmes que pour l'attribut Réinitialiser le type.

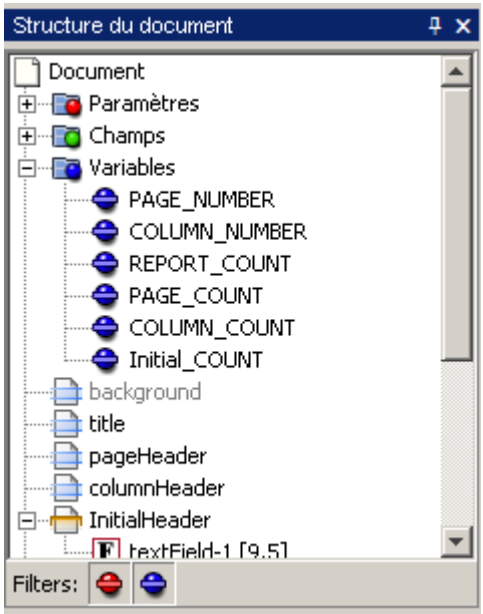
Nous souhaitons maintenant afficher en pied de groupe de la liste d'employés regroupés par initiale, le nombre d'employés pour chaque groupe.

Plusieurs possibilités vont s'offrir à nous :

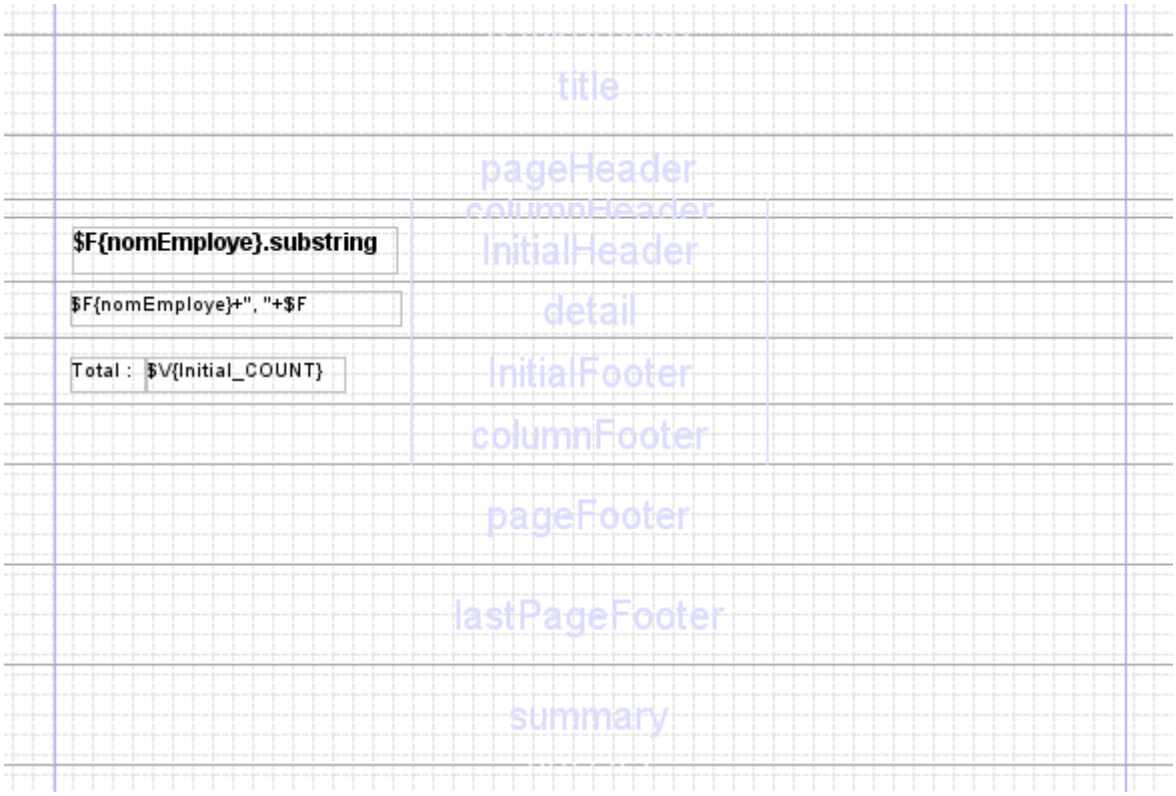
- Utiliser la variable intégrée Initial_COUNT, associée à notre groupe Initial : celle-ci est la meilleure réponse à notre besoin.
- Utiliser la variable intégrée COLUMN_COUNT : en effet, comme chacun de nos groupes est placé dans une colonne différente, cette variable répondrait également à notre besoin
- Utiliser une variable défini par l'utilisateur : la création d'une variable utilisant les attributs offerts par JasperReports (un type de calcul Décompte, avec une réinitialisation sur la groupe Initial) répondrait encore à notre besoin.



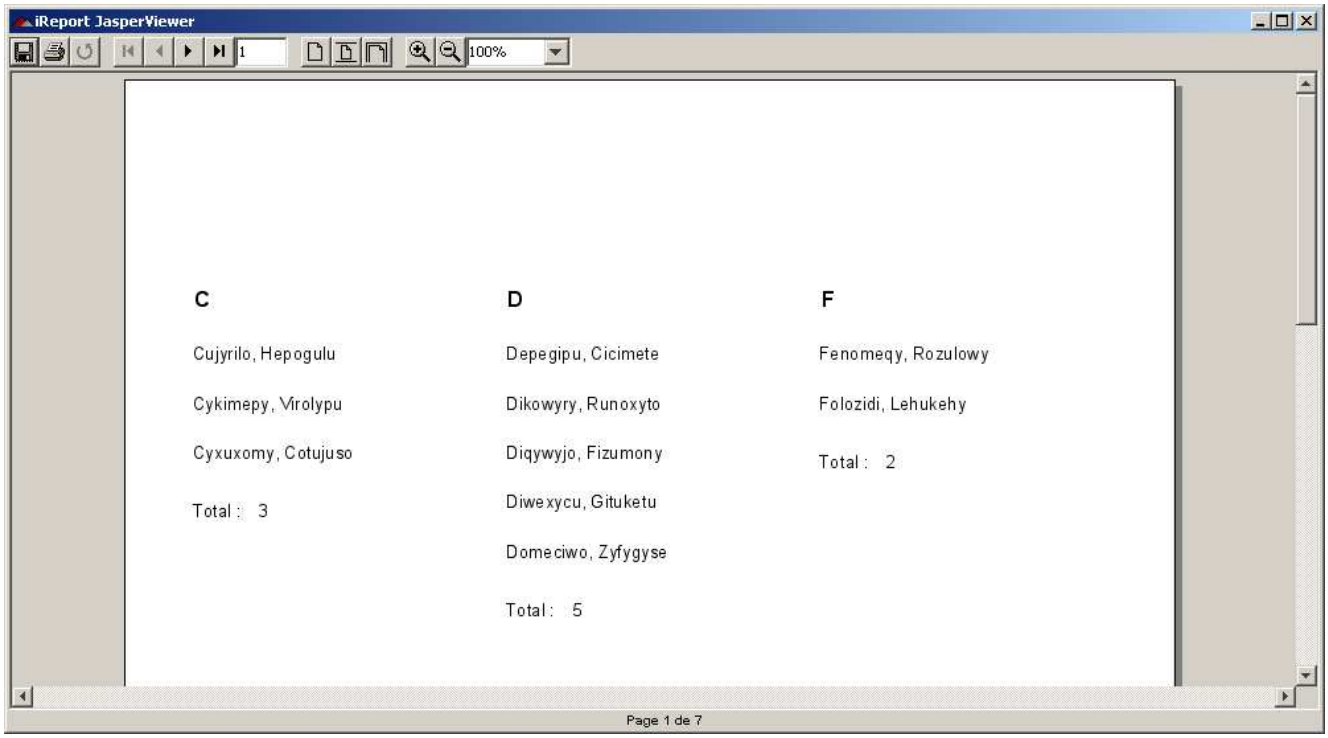
Les variables du rapport sont listés dans la fenêtre Structure du document de iReport :



Le modèle de rapport est modifié pour afficher le total des employés de chaque groupe, en utilisant la variable Initial_COUNT :



Pour le résultat suivant :



5.4 SOUS RAPPORT

Les sous rapports permettent l'incorporation d'un rapport à l'intérieur d'un autre rapport. Les avantages de cette fonctionnalité sont nombreux, notamment la possibilité de garder des rapports au contenu et à la mise en page simple pour composer un rapport potentiellement complexe. Les possibilités de factorisation d'un élément de rapport présent dans plusieurs rapports de l'application est également à noter.

Nous avons vu que pour remplir un rapport, 3 éléments sont nécessaires :

- Un modèle de rapport
- Des paramètres
- Une datasource (ou une connexion JDBC)

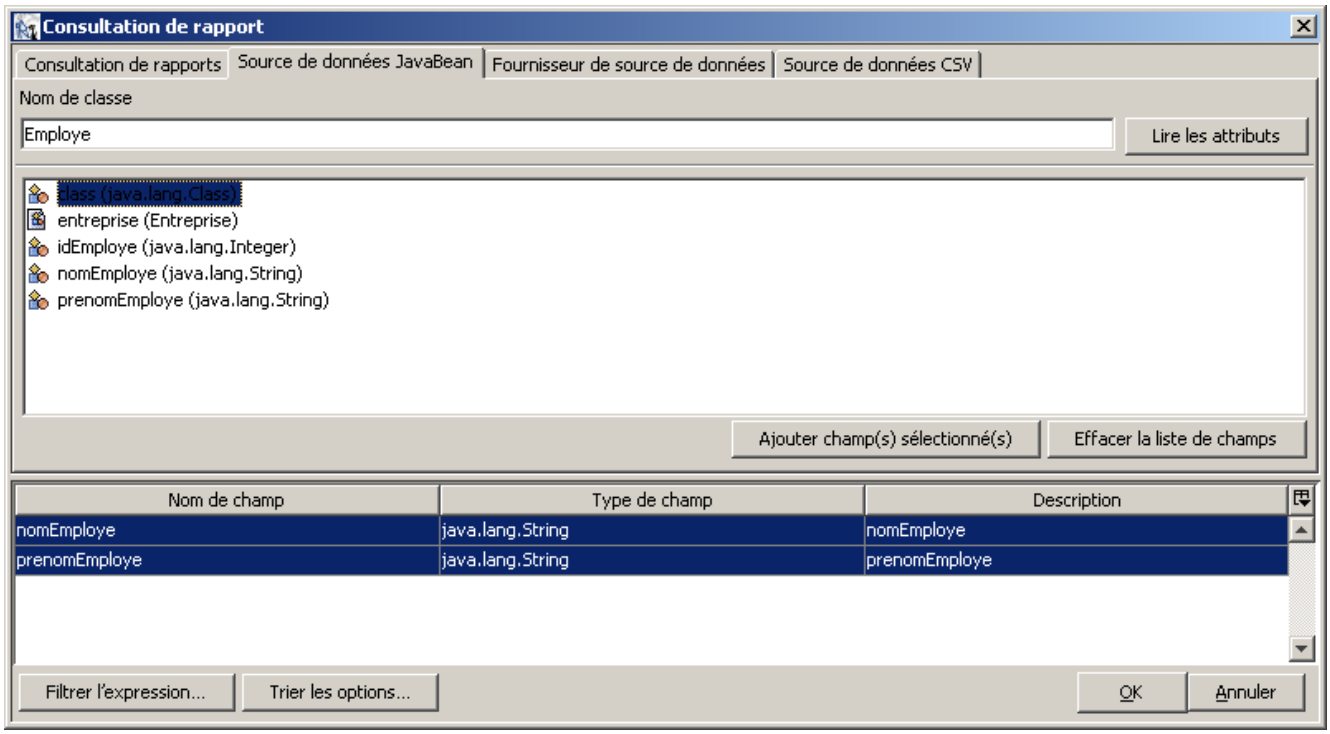
Ces 3 éléments devront donc être indiqué à l'élément sous rapport, via le rapport parent.

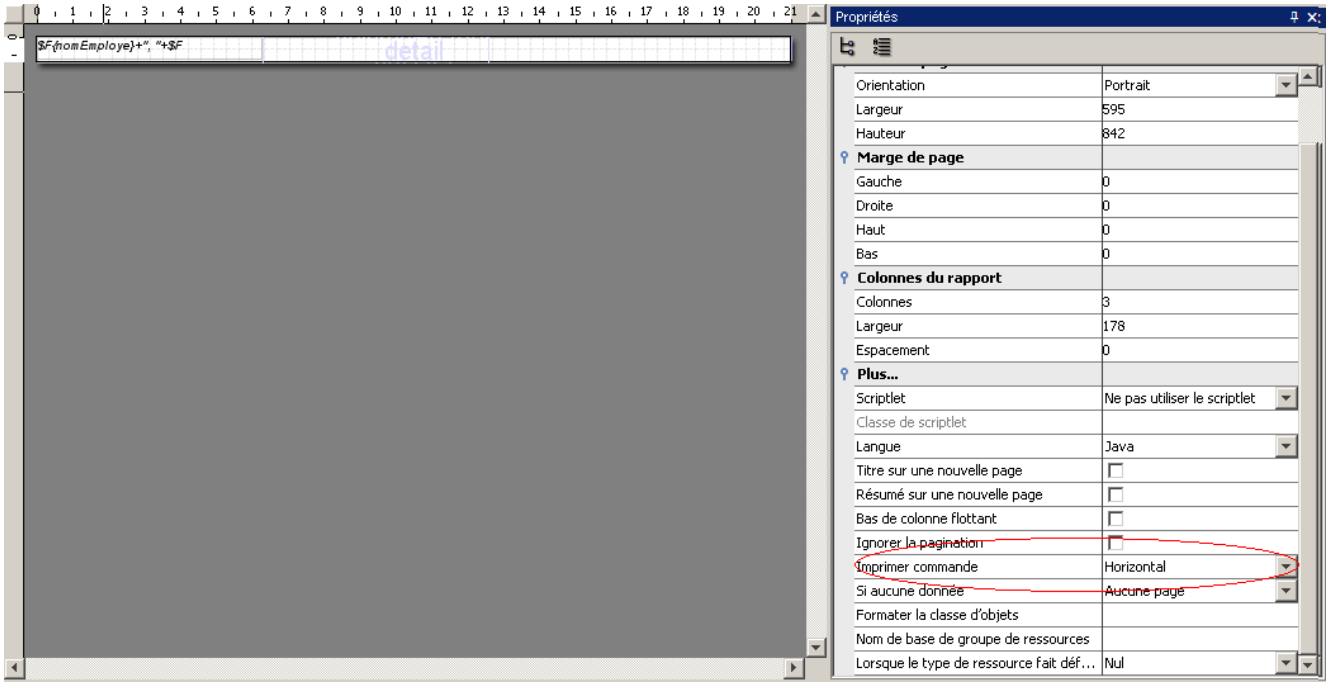
Pour illustrer l'utilisation d'un sous rapport, nous allons reprendre le premier exemple, la liste des projets (cf §4). Nous allons ajouter à cet exemple, pour chaque projet listé, la liste des employés travaillant sur ce projet. Cette liste sera un sous rapport.

Commençons par créer ce sous rapport.

Ce rapport utilisera une datasource lui fournissant une collection d'employés. Cette datasource sera créée à l'exécution par le rapport principal et passée au sous rapport.

Nous créons un rapport utilisant les champs de la classe Employe, en affichant les enregistrements sur 3 colonnes, avec remplissage horizontal :

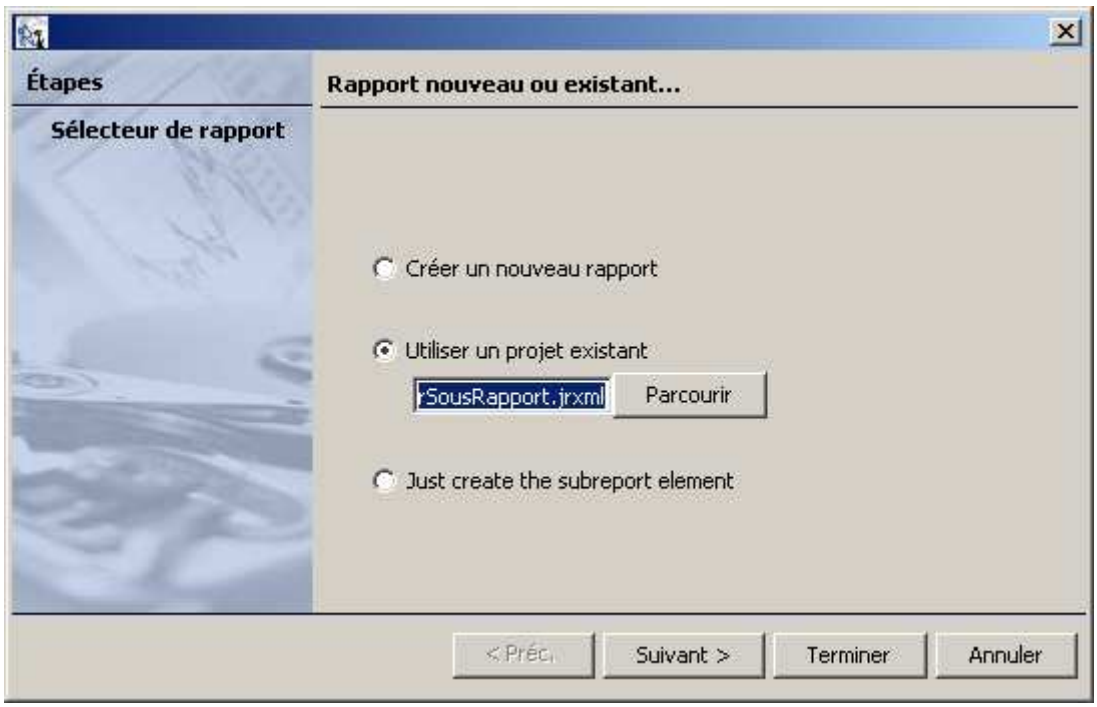




Intégrons maintenant ce rapport dans notre rapport principal, par l'intermédiaire d'un élément sous rapport.

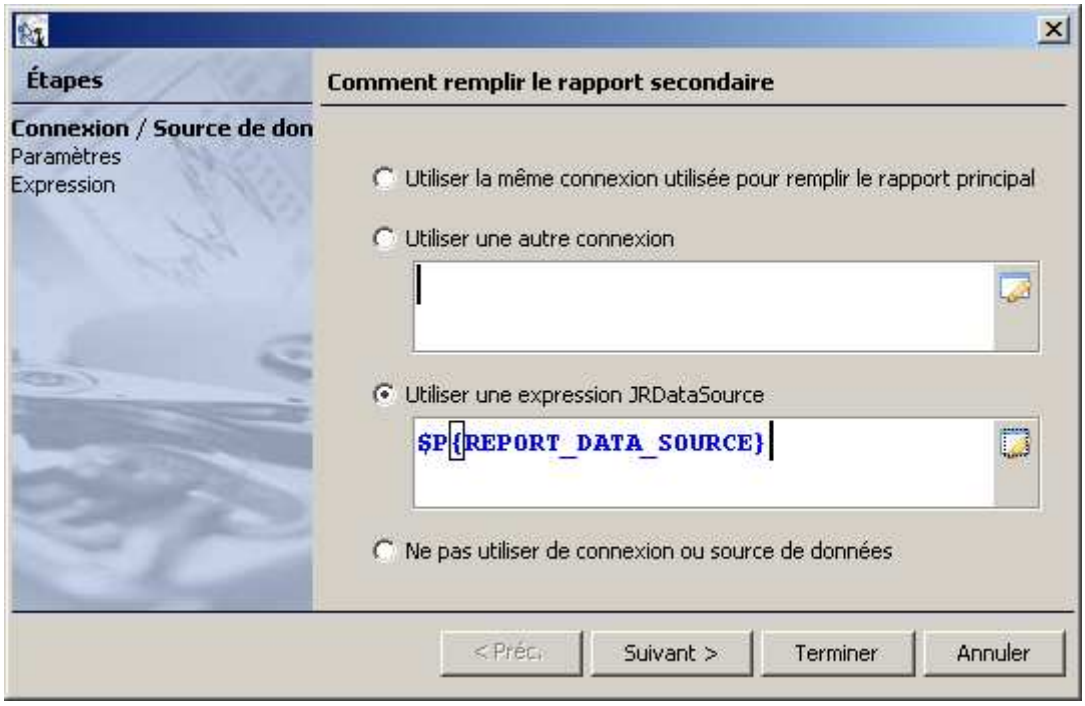
IReport propose un assistant, accessible via l'icône  .

Via l'assistant, nous indiquons que nous utilisons en tant que sous rapport un rapport existant :



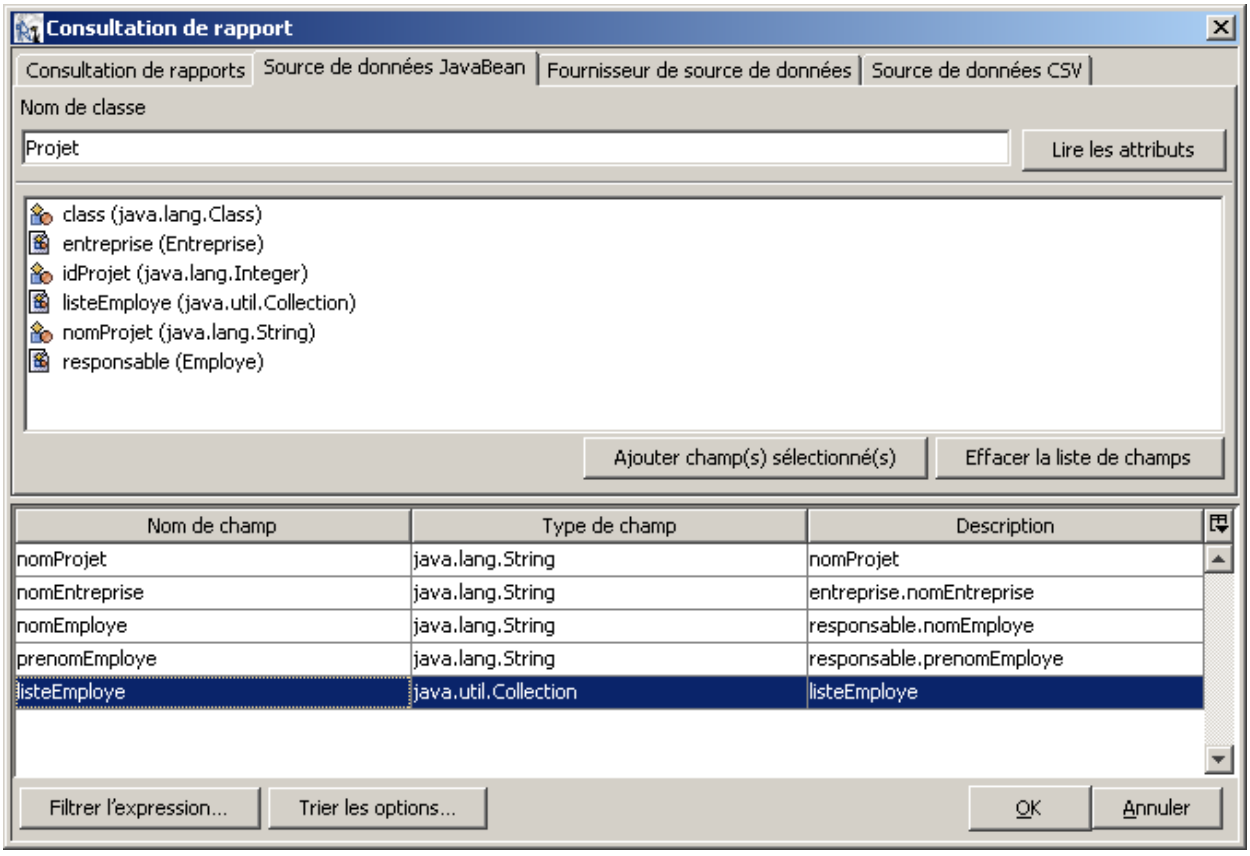
Il faut ensuite indiquer quelle datasource (ou connexion) va utiliser le sous rapport. Les possibilités ici sont diverses :

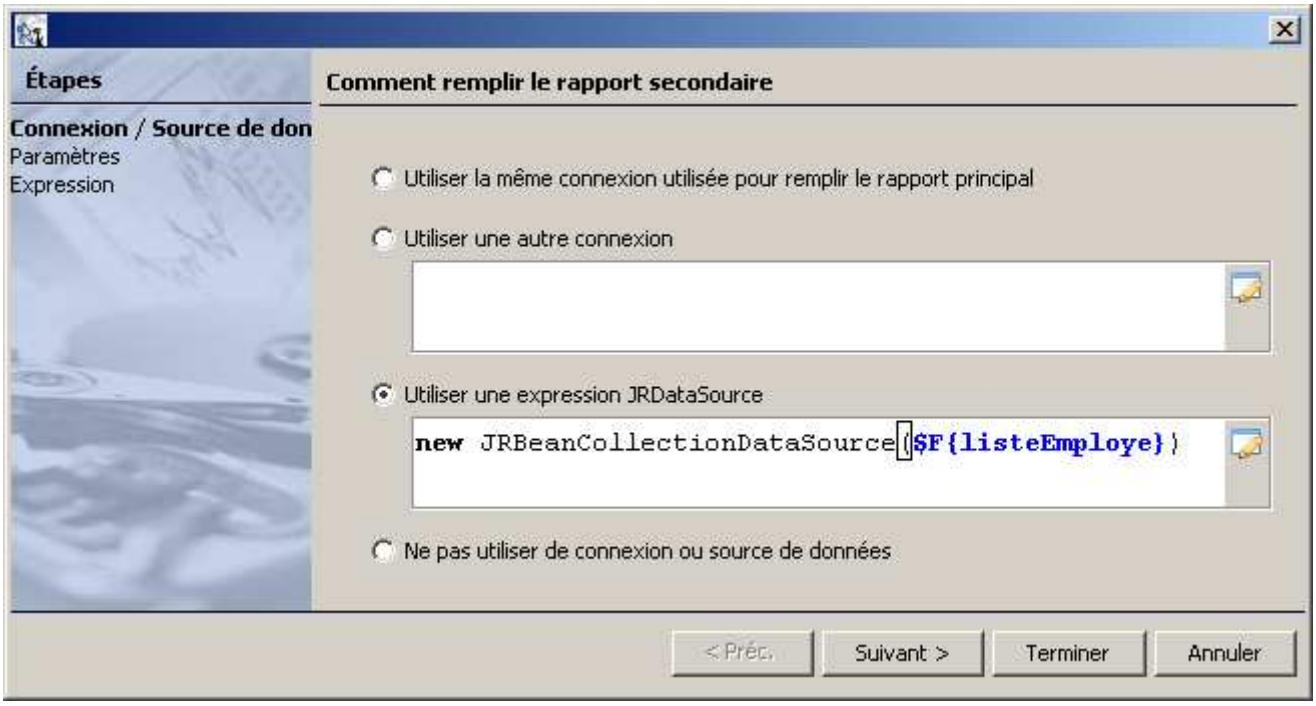
- Le sous rapport pourra utiliser la même datasource (ou connexion) que le rapport principal, que celui-ci pourra lui passer via les paramètres intégrés REPORT_DATA_SOURCE (ou REPORT_CONNECTION)



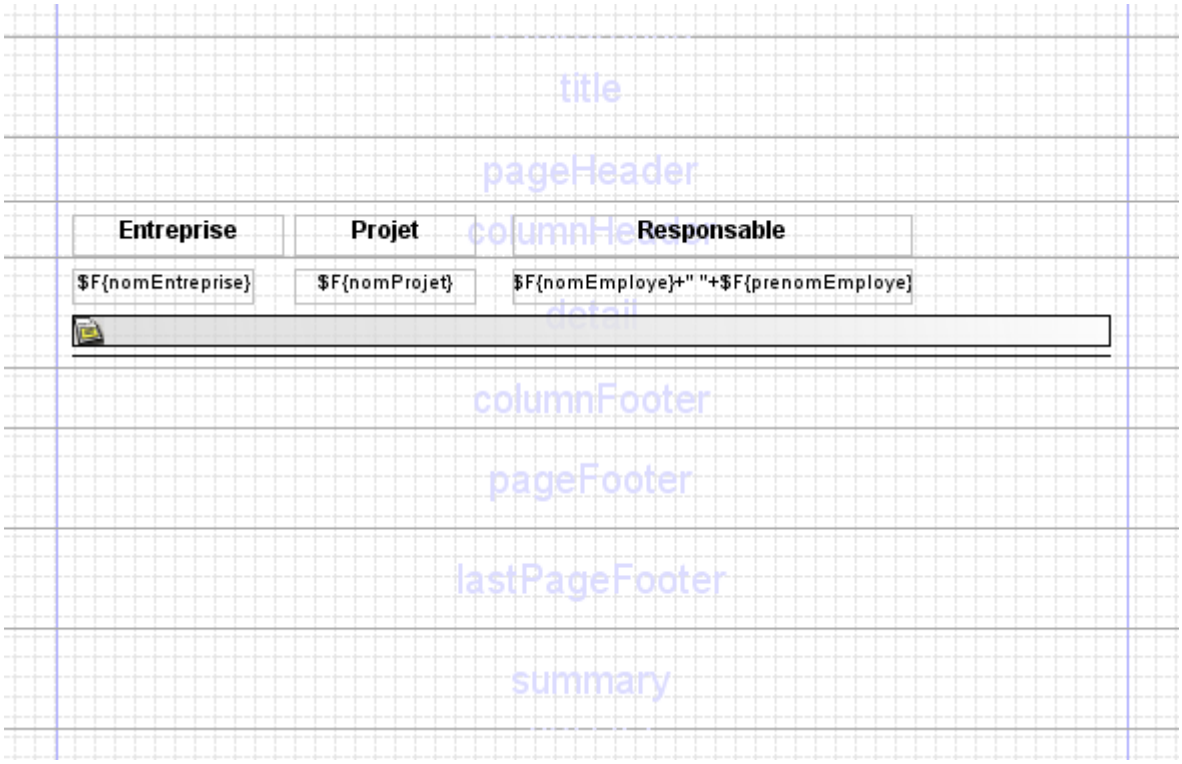
- Le sous rapport pourra utiliser une autre datasource (ou connexion) que le rapport principal pourra lui passer via un paramètre utilisateur renseigné à l'exécution du rapport principal.

- Le sous rapport peut enfin utiliser une datasource créée à l'exécution à partir d'un paramètre ou d'un champ du rapport principal. C'est la possibilité que nous allons choisir ici, en créant une instance de la classe JRBeanCollectionDataSource de JasperReports, prenant en paramètre la collection d'instance de la classe Employe portée par la classe Projet, que nous avons incluse dans les champs du rapport principal.



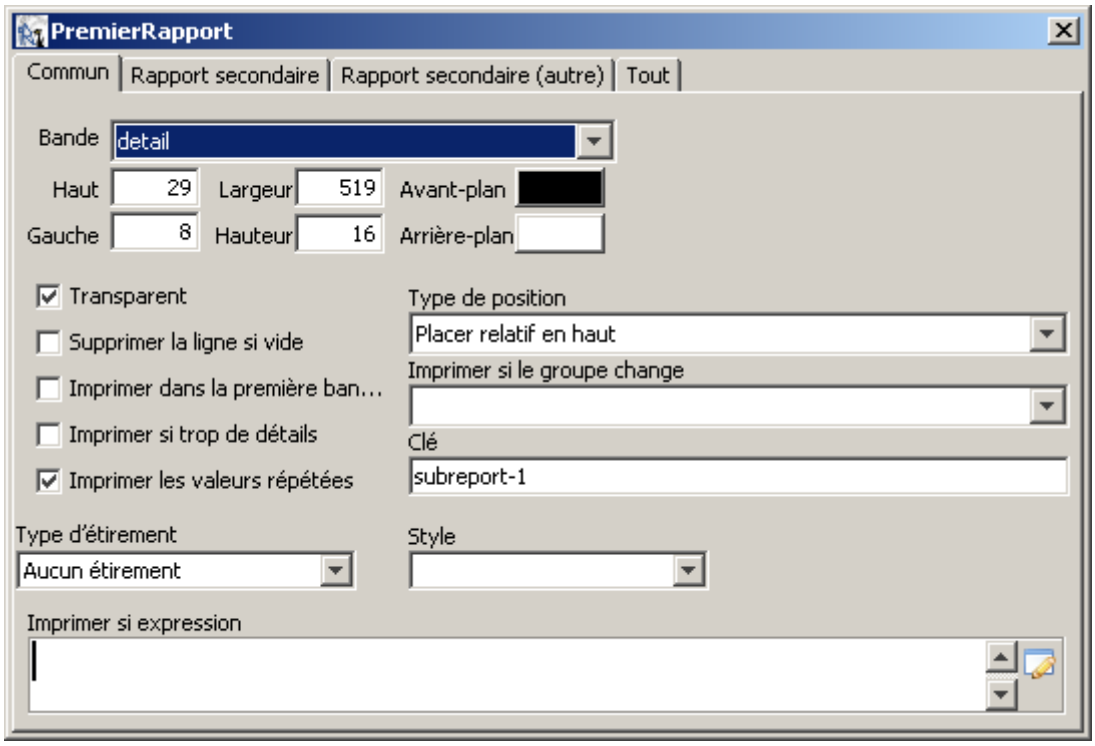


L'élément sous rapport créé apparaît dans le modèle de notre rapport principal :

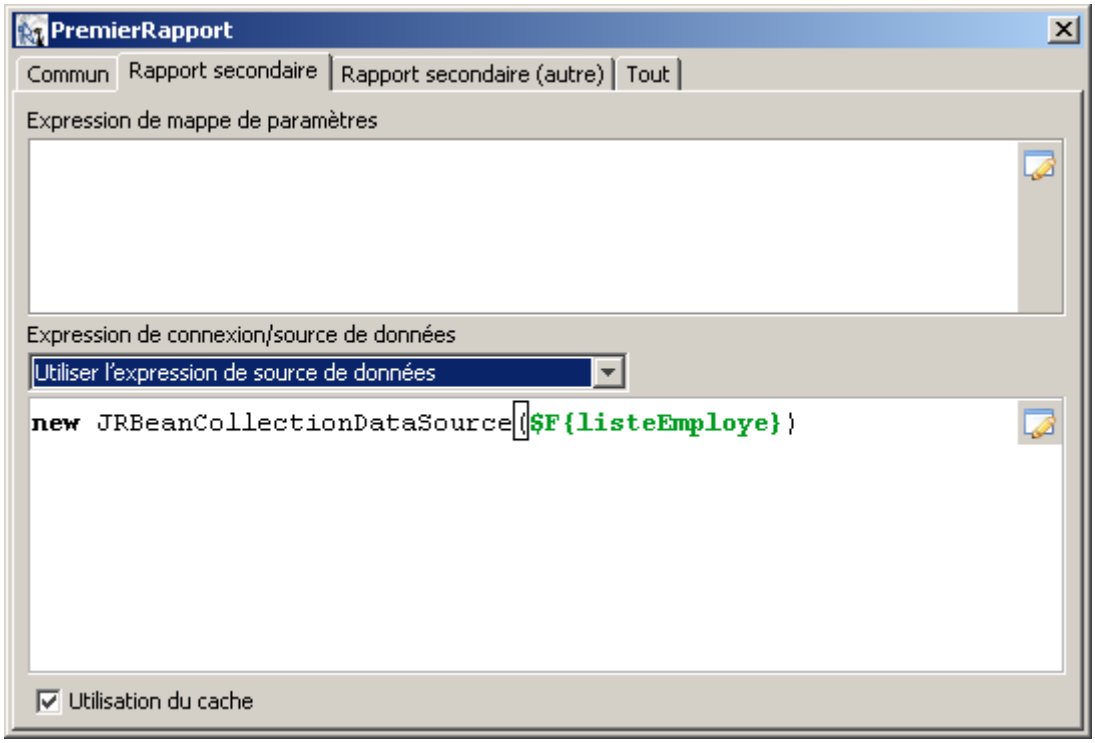


En double cliquant sur cet élément, on accède à la fenêtre de paramétrage du sous rapport, que nous allons détailler :

- L'onglet Commun permet de régler la mise page du sous rapport dans le rapport principal, en particulier de préciser une éventuelle condition d'affichage, sous la forme d'une expression



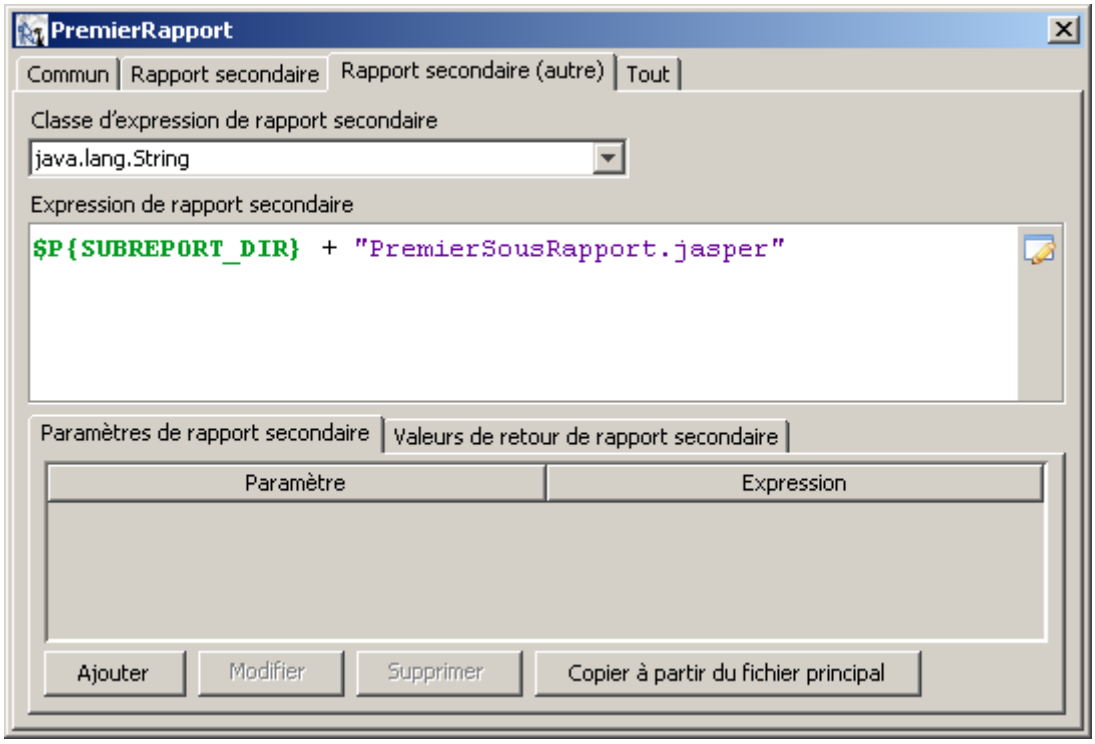
- L'onglet Rapport secondaire permet de définir une Map de paramètres à passer au sous rapport (peu utilisé), ainsi que l'expression définissant la datasource (ou connexion) à fournir au sous rapport



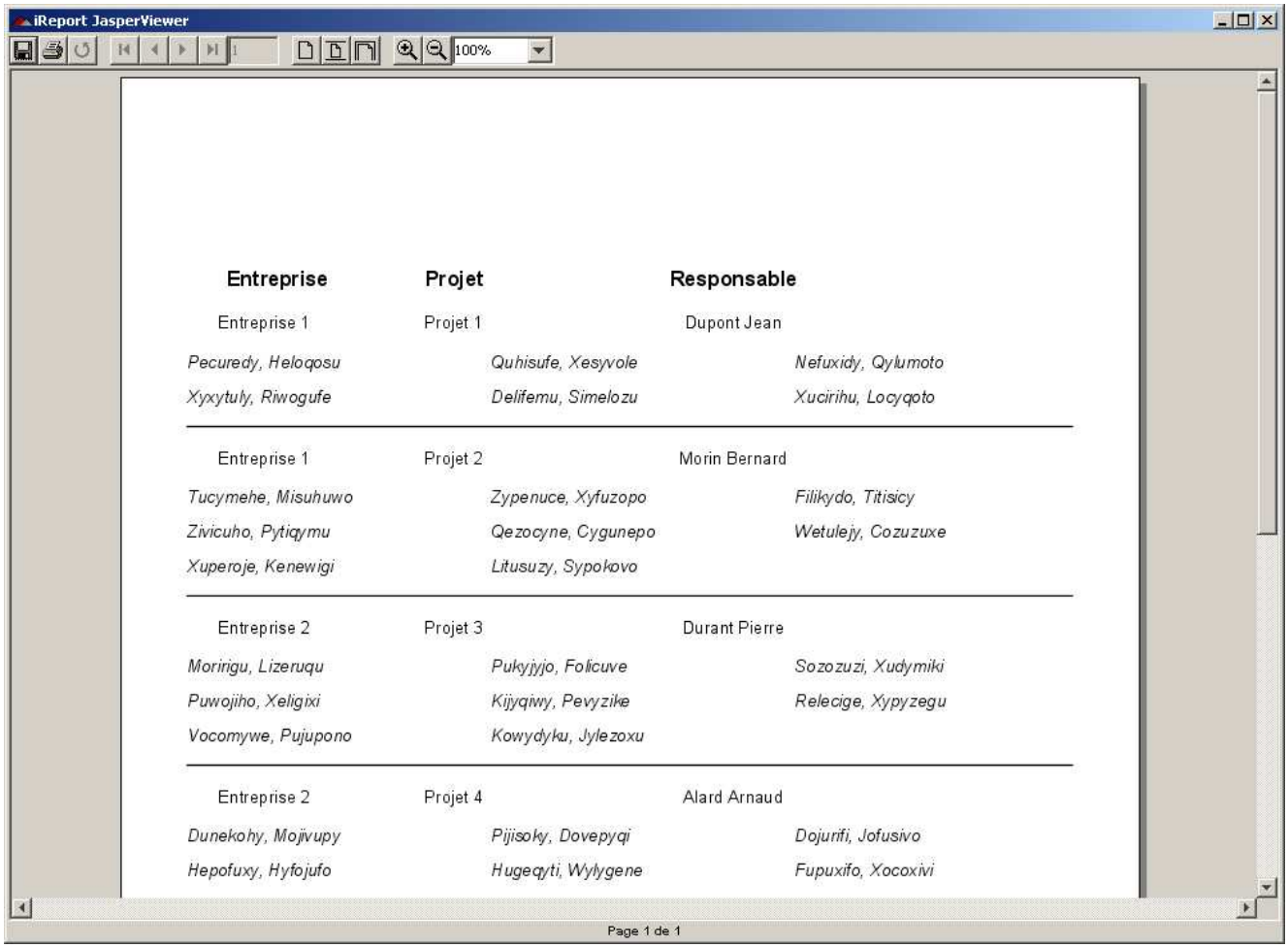
- L'onglet Rapport secondaire (autre) permet de définir l'accès au rapport qui servira de sous rapport. Cet accès peut être déclaré de plusieurs manières :
    - Sous la forme d'un String. C'est la manière la plus commune. Dans ce cas, JasperReports va d'abord essayer de se servir de cette chaîne comme d'une URL. Si cela échoue, la String sera utiliser comme chemin d'accès sur le système de fichier local. Si cela échoue encore, JasperReports essaiera de charger le modèle de rapport depuis le CLASSPATH. Si tout ceci échoue, une Exception est levée.
    - Sous la forme d'une instance de la classe `JasperReport` de JasperReports.
    - Sous la forme d'une instance de la classe `java.io.File`
    - Sous la forme d'une instance de la classe `java.io.InputStream`
    - Sous la forme d'une instance de la classe `java.net.URL`

  Pour ces 4 dernières formes, la façon la plus commune de passer ces instances est de passer par un paramètre du rapport.

  Enfin cet onglet permet de mapper les paramètres du sous rapport avec des éléments (paramètre, variable, champ ou expression) du rapport principal ; ainsi que, inversement, de mapper des variables du rapport secondaire avec des variables du rapport principal pour fournir un moyen au rapport secondaire de retourner des valeurs au rapport principal.



L'exécution du nouveau rapport principal donne le résultat suivant :



L'élément sous rapport a la forme suivante dans le code source du rapport principal :

```
<subreport  isUsingCache="true">
        <reportElement
                x="8"
                y="29"
                width="519"
                height="16"
                key="subreport-1"/>
        <dataSourceExpression>
                <![CDATA[new JRBeanCollectionDataSource($F{listeEmploye})]]>
        </dataSourceExpression>
        <subreportExpression  class="java.lang.String">
                <![CDATA[$P{SUBREPORT_DIR} + "PremierSousRapport.jasper"]]>
        </subreportExpression>
</subreport>
```

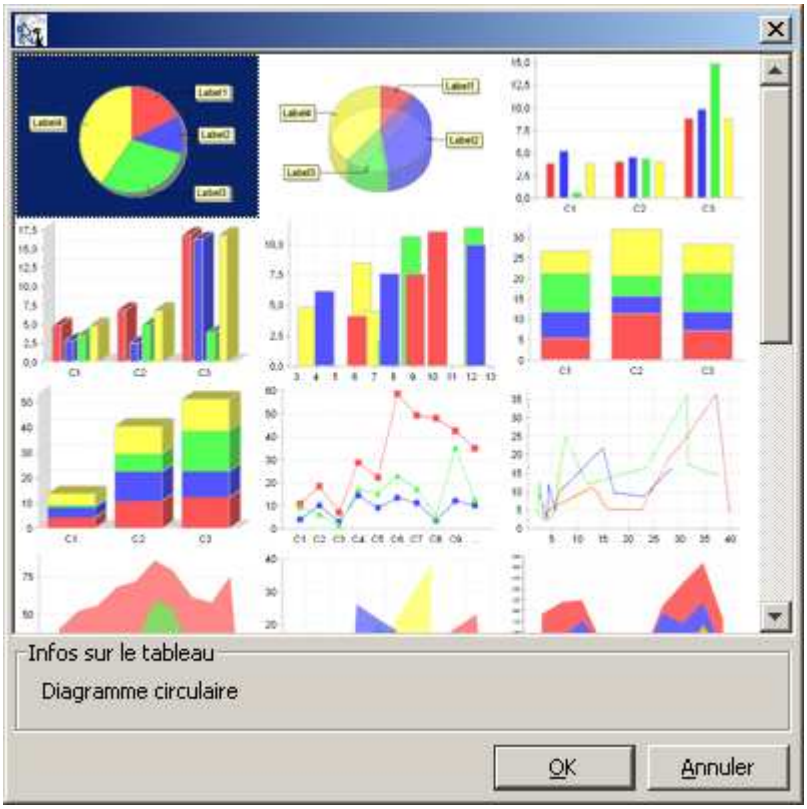
5.5 GRAPHIQUES

JasperReports propose la possibilité de représenter les données sous forme de graphique. Cette fonctionnalité se base sur l'utilisation de la bibliothèque Java JFreeChart. La gamme de diagramme proposés est large :

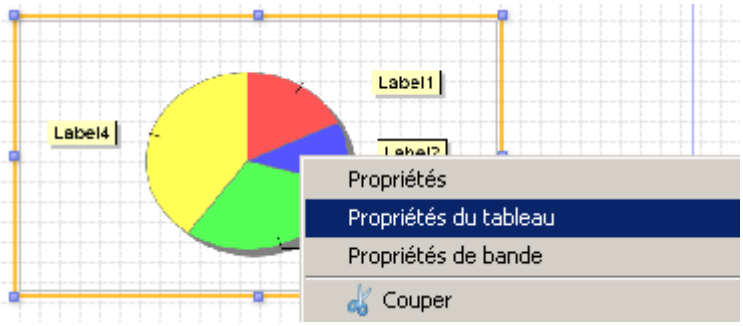
- Camenbert
- Barres
- Courbe
- …

Chaque type de diagramme s'appuiera sur un ensemble de données défini par l'utilisateur. Voyons ceci par l'exemple en affichant sur le rapport précédent un camembert représentant le nombre d'employés par projet.

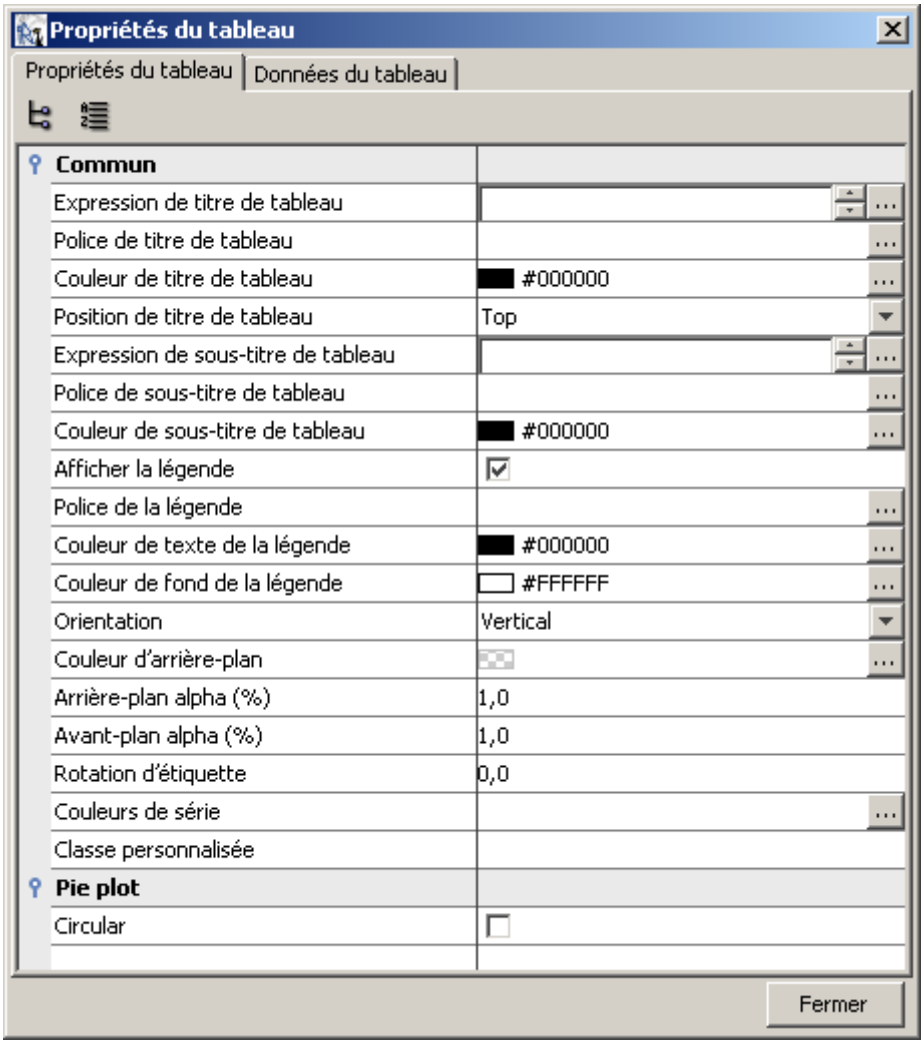
Dans iReport, l'ajout d'un diagramme dans le rapport se fait par clic sur l'icône  puis en choisissant le type de diagramme à rajouter :



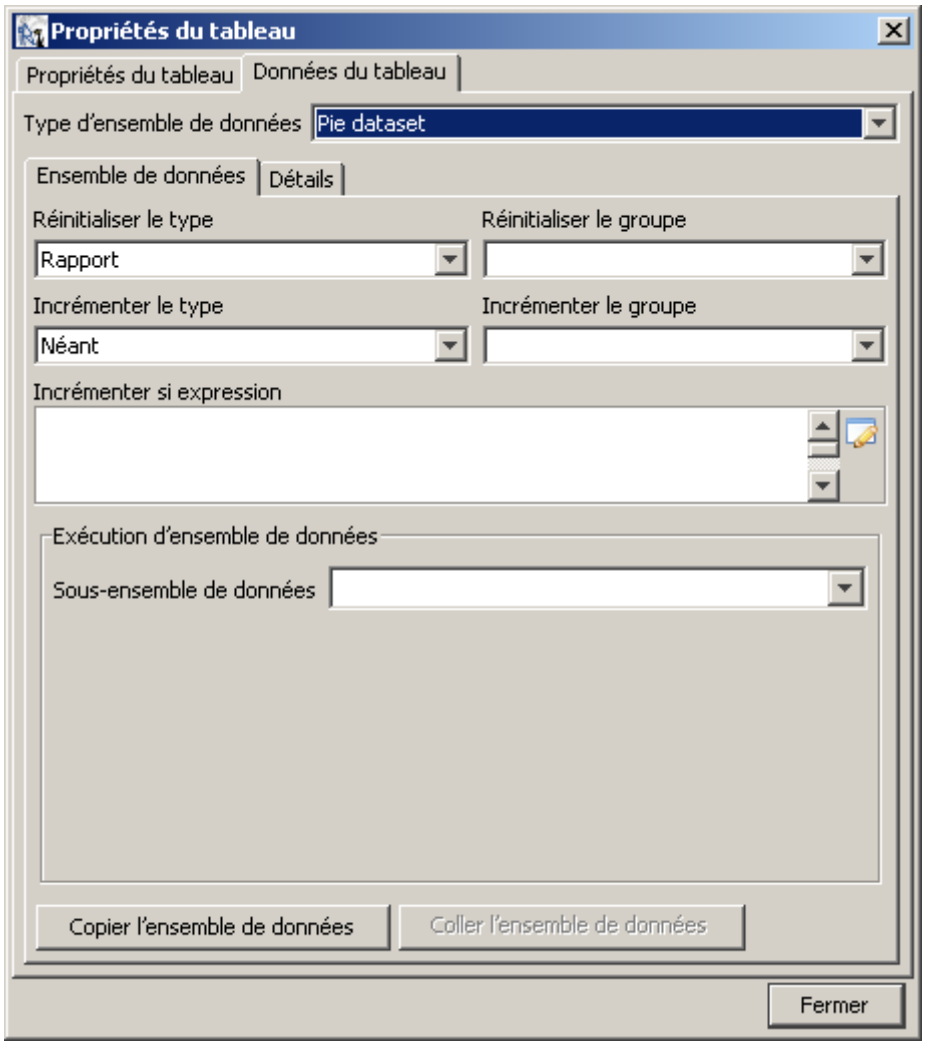
L'accès aux propriétés du diagramme se fait en faisant apparaître le menu contextuel de l'élément (clic-droit sur l'élément) et en sélectionnant Propriétés du tableau :



L'onglet Propriétés du tableau contient une liste de paramètres contrôlant l'apparence du diagramme.

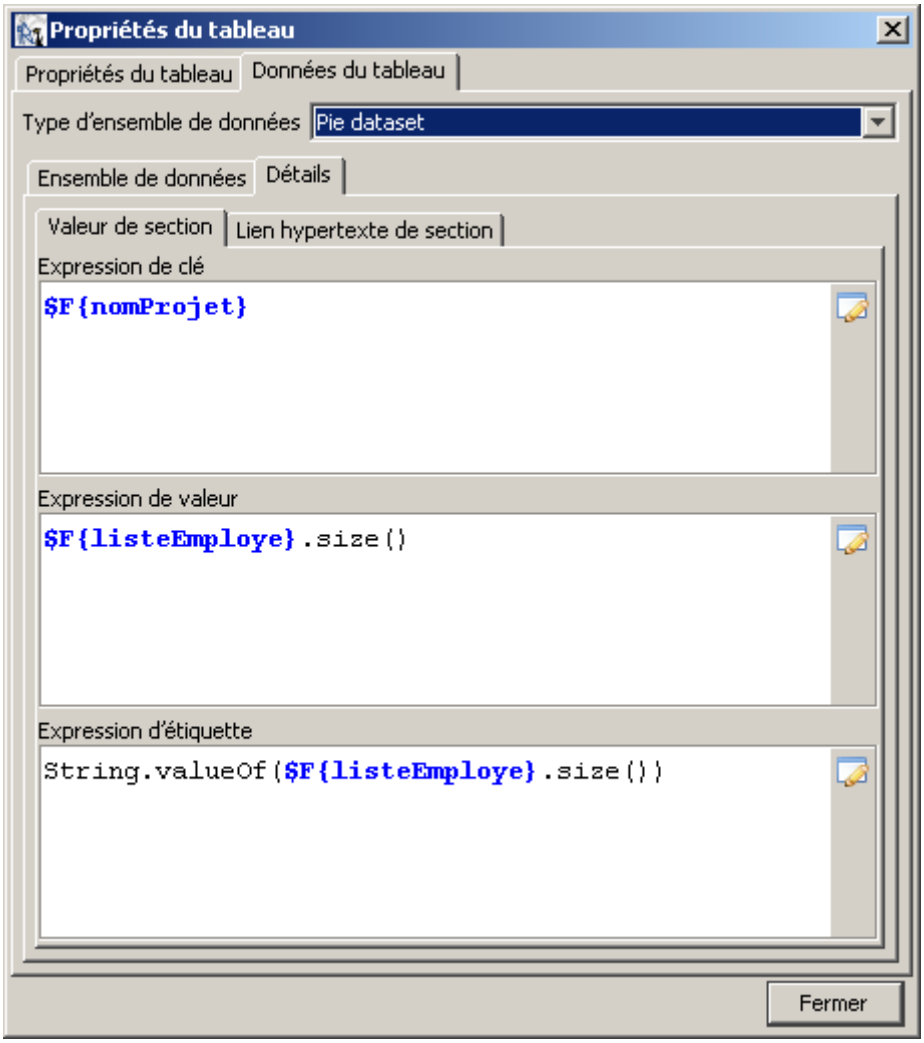


L'onglet Données du tableau va permettre de définir les données définissant le diagramme.



Le sous onglet Ensemble de données règle les paramètres de réinitialisation et d'incrémentation de la valeur de l'expression du rapport.

Enfin, le sous onglet Détails permet de définir l'ensemble de données sur lequel le diagramme va s'appuyer. Cet onglet sera différent en fonction du type de diagramme choisi, afin de définir un ensemble de données en adéquation avec le diagramme.



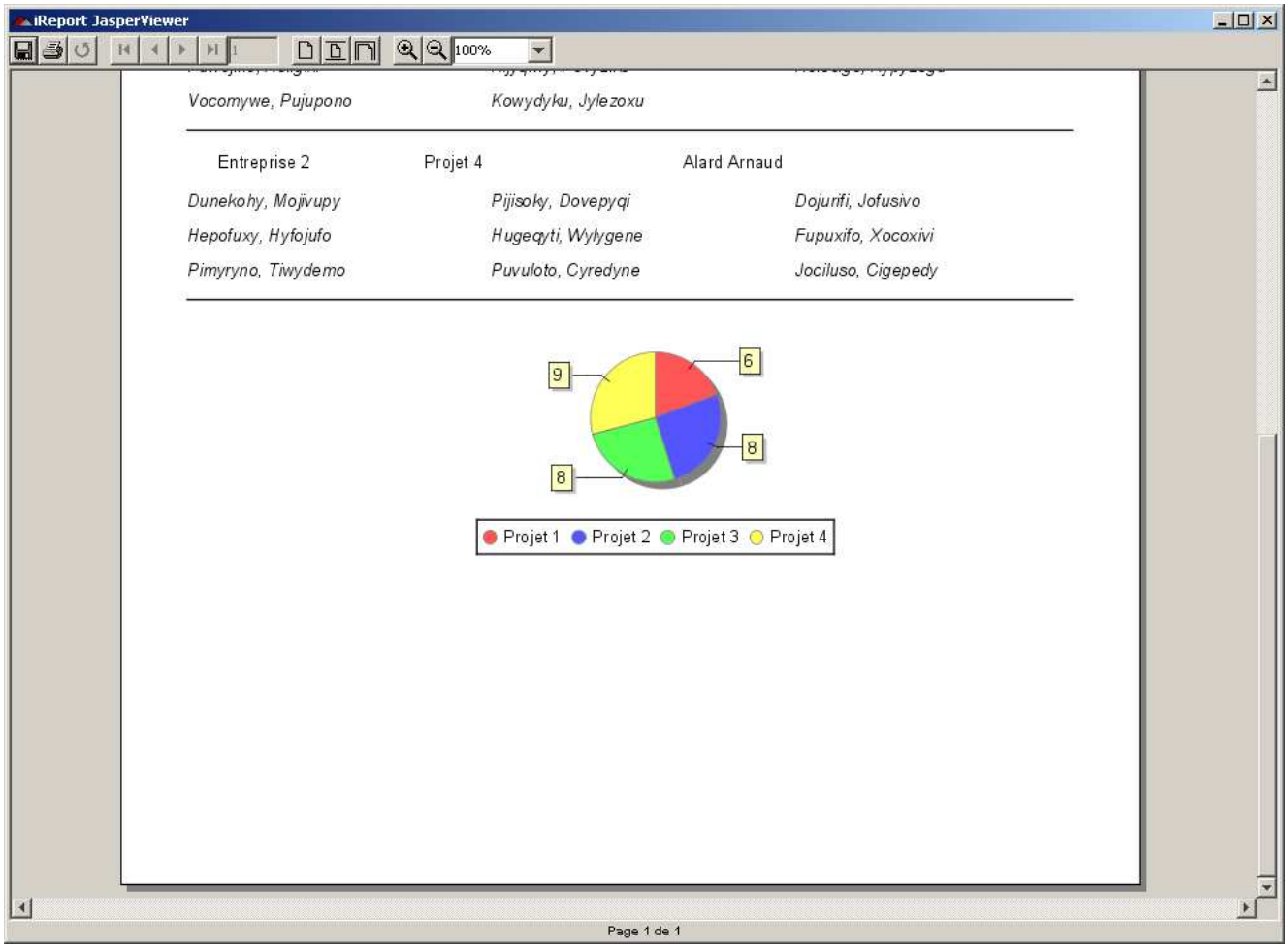
Dans notre exemple, pour un diagramme de type camembert, l'ensemble de données du diagramme va être définir par l'intermédiaire de 3 éléments :

- L'expression de clé permet d'identifier de manière unique chaque part du camembert.
- L'expression de valeur spécifie la valeur numérique à associer avec la clé
- L'expression d'étiquette spécifie le label associé à chaque part du camembert

Notre exemple désirant le nombre d'employés par projet, les valeurs suivantes sont utilisées :

- Clé : le nom, unique, du projet
- Valeur : la taille de la collection d'employés du projet
- Etiquette : cette même taille, mais convertit en chaîne de caractères

Le diagramme obtenu correspond à nos attentes :



La représentation de ce diagramme dans le code source du modèle de rapport est la suivante :

```
<pieChart>
	<chart  hyperlinkTarget="Self" >
		<reportElement
			x="8"
			y="8"
			width="239"
			height="135"
			key="element-1"/>
		<box></box>
		<chartLegend textColor="#000000" backgroundColor="#FFFFFF" >
		</chartLegend>
	</chart>
	<pieDataset>
		<dataset></dataset>
		<keyExpression><![CDATA[$F{nomProjet}]]></keyExpression>
		<valueExpression>
			<![CDATA[$F{listeEmploye}.size()]]>
```

```
                </valueExpression>
                <labelExpression>
                        <![CDATA[String.valueOf($F{listeEmploye}.size())]]>
                </labelExpression>
                <sectionHyperlink></sectionHyperlink>
        </pieDataset>
        <piePlot>
                <plot />
        </piePlot>
</pieChart>
```

5.6 SCRIPTLETS

JasperReports autorise l'exécution de portions de code Java à certains moments du remplissage d'un rapport. C'est portions de code sont appelés scriptlets.

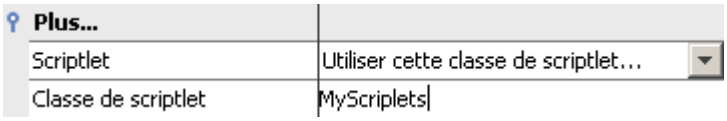
L'écriture se réalise en entendant une des deux classes de JasperReports suivantes :

- `JRAbstractScriplet` : cette classe contient un certain nombre de méthodes abstraites qui devront être surchargées dans la classe fille
- `JRDefaultScriplet` : cette classe utilitaire étend `JRAbstractScriplet` et fournit une implémentation par défaut vide de chacune des méthodes de la classe mère. En héritant de cette classe, l'utilisateur a seulement besoin de surcharger les méthodes dont il a besoin.

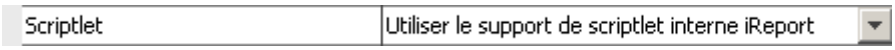
Les méthodes définies par la classe JRAbstractScriplet sont automatiquement appelées par JasperReports à un moment précis lors du remplissage du rapport. Ces méthodes, au nom révélateur, sont les suivantes :

- `void beforeReportInit()`
- `void afterReportInit()`
- `void beforePageInit()`
- `void afterPageInit()`
- `void beforeColumnInit()`
- `void afterColumnInit()`
- `void beforeGroupeInit()`
- `void afterGroupeInit()`
- `void beforeDetailEval()`
- `void afterDetailEval()`

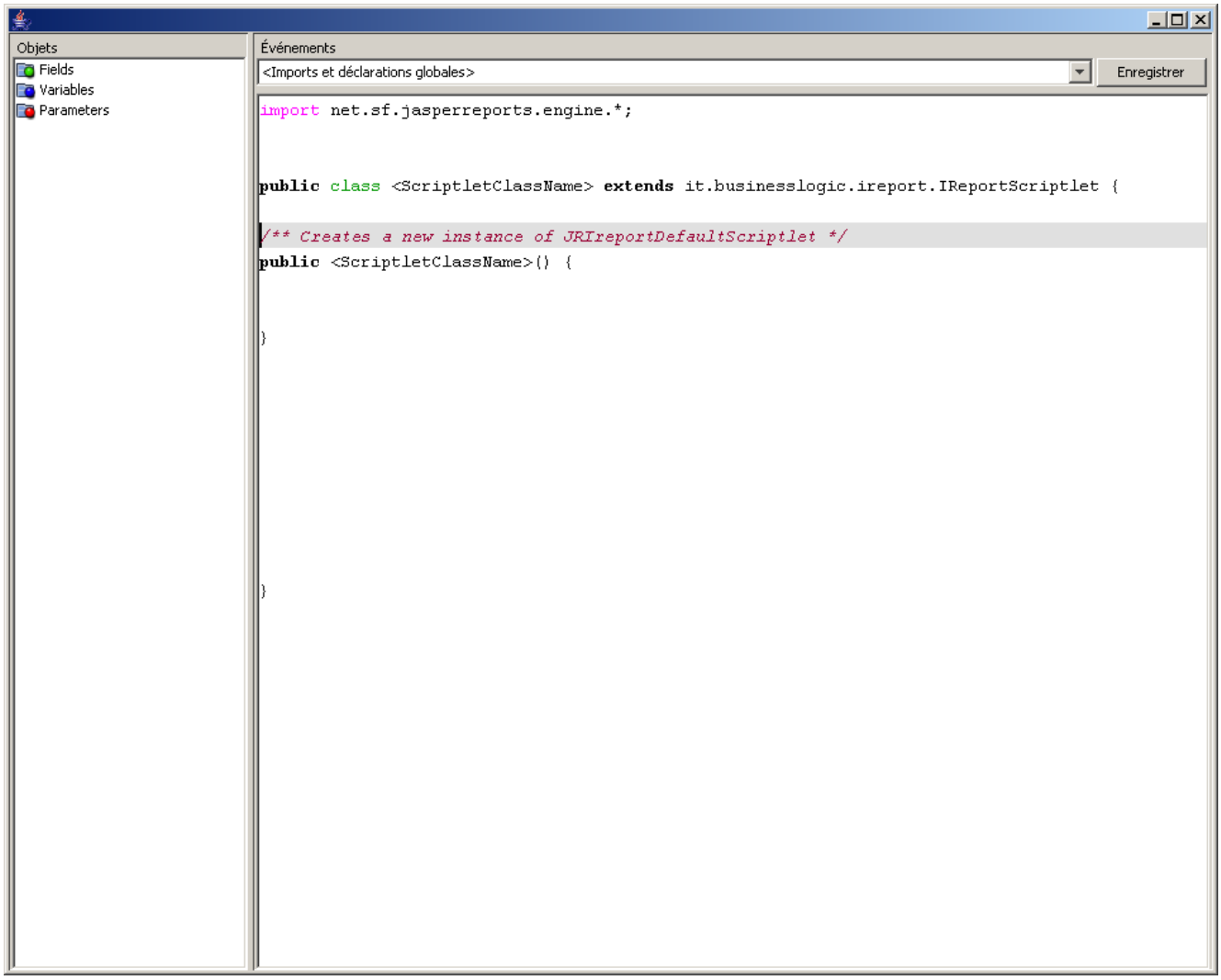
La classe de scriptlets de l'utilisateur peut être écrite et compiler via un outil externe comme Eclipse et le chemin vers cette classe précisé dans les options du document :



iReport propose également un éditeur de scriptlets. En utilisant ce mode, la classe Java de scriptlets est géré en interne par iReport.



L'éditeur de scriptlets de iReport est accessible par le menu Modifier > Editeur de scriptlets.



Les scriptlets ont accès aux variables, champs et paramètres du rapport, par l'intermédiaire des méthodes suivantes :

- `getVariableValue(`*`nomVariable`*`)` et `setVariableValue(`*`nomVariable, valeur`*`)` pour récupérer et modifier la valeur d'une variable
- `getFieldValue(`*`nomChamp`*`)` et `setFieldValue(`*`nomChamp, valeur`*`)` pour récupérer et modifier la valeur d'un champ
- `getParameterValue(`*`nomParametre`*`)` et `setParameterValue(`*`nomParametre, valeur`*`)` pour récupérer et modifier la valeur d'un paramètre

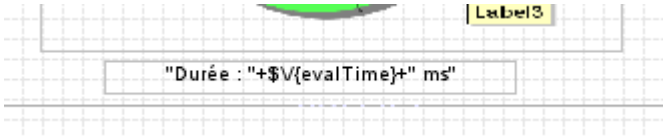
Notons enfin que la classe de scriptlets de l'utilisateur pourra contenir d'autres méthodes que celles définies par la classe `JRAbstractScriplet`, et que ces méthodes pourront être appelées par l'intermédiaire du paramètre intégré REPORT_SCRIPTLET.

Par exemple si la classe contient une méthode toto(), elle pourra être appelée par la syntaxe :

```
$P{REPORT_SCRIPTLET}.toto()
```

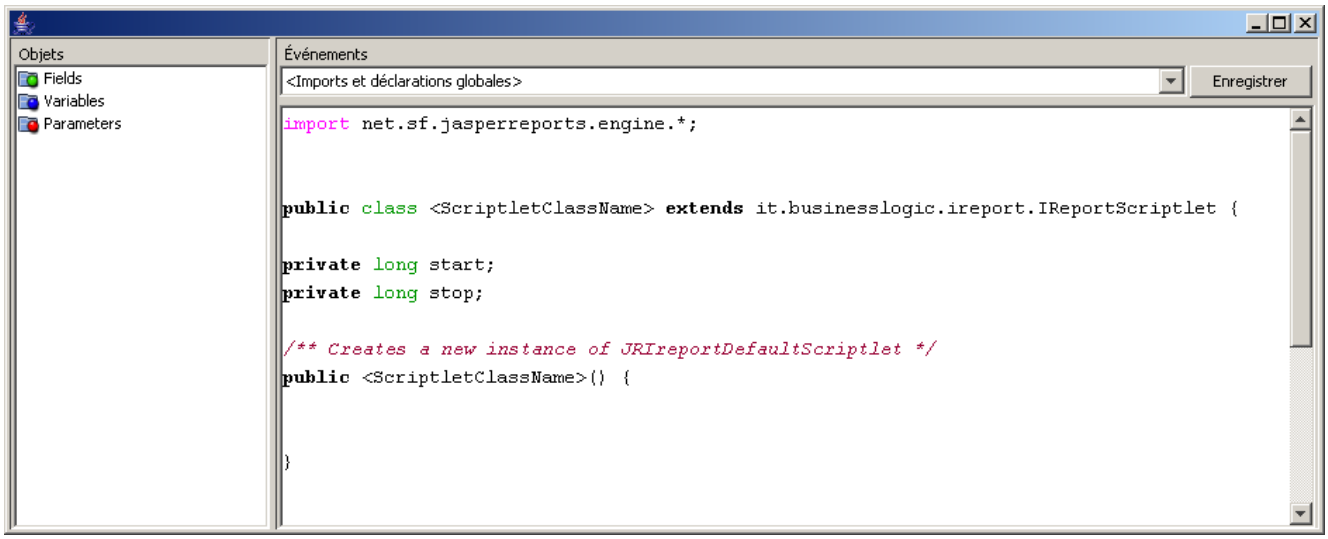
Illustrons l'utilisation de scriptlets en affichant sur notre rapport le temps d'exécution entre l'initialisation du rapport et la fin de l'évaluation des données.

Nous modifions le rapport pour afficher une variable nommée evalTime qui aura été préalablement créée :

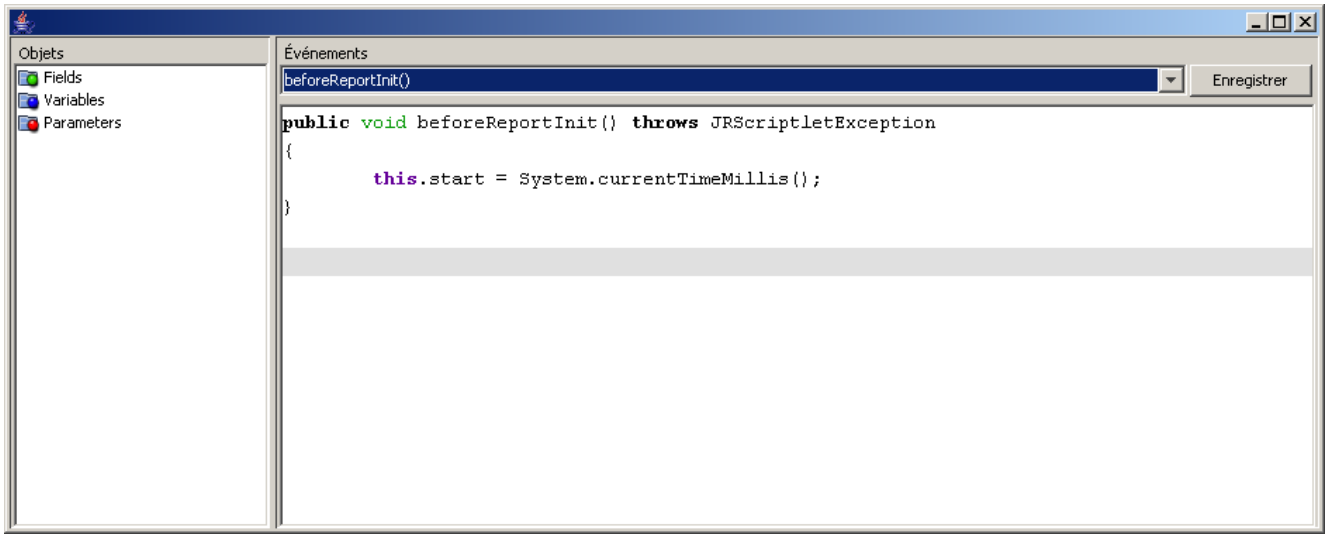


Via l'éditeur de scriptlets de iReport, nous renseignons les éléments suivants :

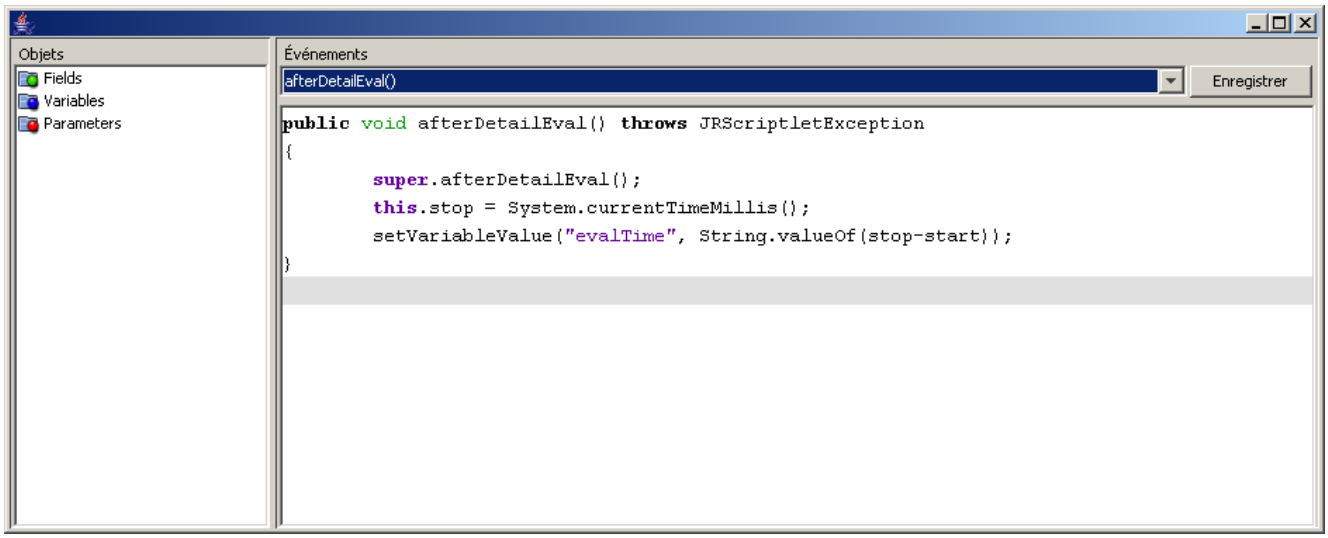
- Deux variables pour la date de début et la date de fin



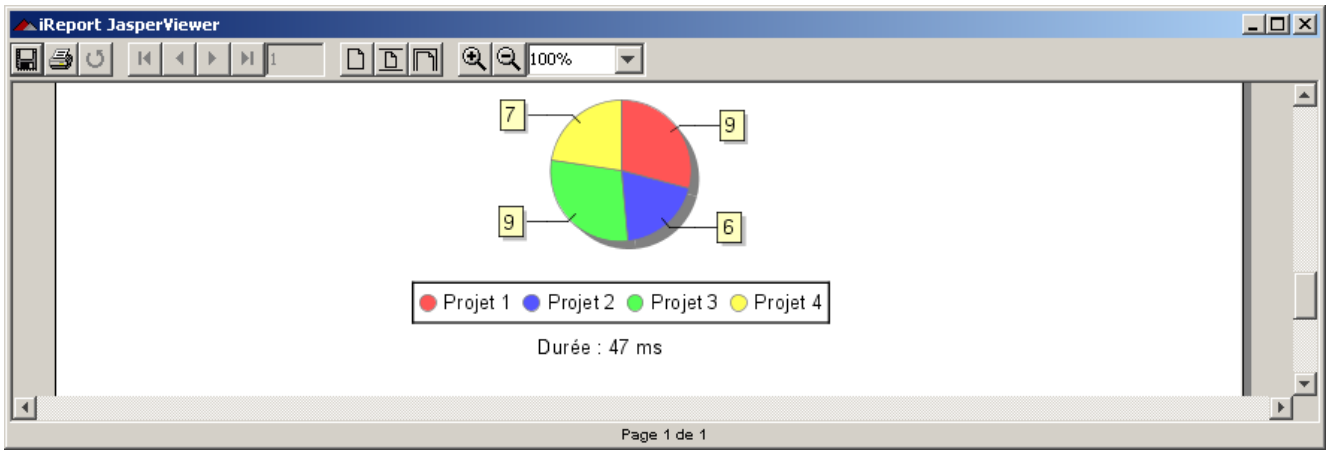
- L'initialisation de la date de début, avant l'initialisation du rapport :



- L'initialisation de la date de fin, puis l'inscription de la durée dans la variable evalTime, après l'évaluation des données :



A l'exécution, nous obtenons le résultat suivant :



Dans le code source du modèle de rapport, l'emploi d'une classe de scriptlets se défini de la manière suivante :

```
<jasperReport
                name="PremierRapport"
                columnCount="1"
                printOrder="Vertical"
                orientation="Portrait"
                pageWidth="595"
                pageHeight="842"
                columnWidth="535"
                columnSpacing="0"
                leftMargin="30"
                rightMargin="30"
                topMargin="20"
                bottomMargin="20"
                whenNoDataType="AllSectionsNoDetail"
                scriptletClass="PremierRapportScriptlet"
                isTitleNewPage="false"
                isSummaryNewPage="false">
```

5.7 VIRTUALISATION

Lors de la génération de très grand rapport, manipulant de grandes quantités de données, il peut arriver que la quantité de mémoire nécessaire à la génération de ce rapport soit supérieure à la quantité de mémoire allouée à la JVM, provoquant une `OutOfMemoryException`.

Pour résoudre ce problème, JasperReports propose un mécanisme de virtualisation de la mémoire : lors de l'exécution du rapport, des segments de données sont stockés sur le disque afin de libérer de la mémoire.

L'utilisation de cette fonctionnalité se fait par l'intermédiaire du paramètre intégré REPORT_VIRTUALIZER. Celui-ci devra être renseigné avec une instance d'une classe implémentant l'interface `JRVirtualizer` de JasperReports. JasperReports propose plusieurs implémentations de cette interface :

- `JRFileVirtualizer` : virtualisation des données dans des fichiers
- `JRSwapFileVirtualizer` : virtualisation des données dans un seul gros fichier
- `JRGzipVirtualizer` : les données non utilisées sont zippées, mais gardées en mémoire

Voici un exemple d'utilisation de `JRFileVirtualizer` dans le code Java :

```
// 2 arguments : le nombre de page maximum en mémoire, le répertoire
// utilisé pour stocké les segments de données
JRFileVirtualizer fileVirtualizer = new JRFileVirtualizer(3, "cache");
HashMap parameters = new HashMap();
parameters.put(JRParameter.REPORT_VIRTUALIZER, fileVirtualizer);
```

En cas de besoin spécifique (si l'on souhaite stocker les segments de données dans une base de données par exemple), il est possible de réaliser une implémentation personnalisée de l'interface `JRVirtualizer`.

Dans iReport, pour utiliser la virtualisation, il faut sélectionner l'option Utiliser le virtualiseur de rapport dans le menu Créer.

Le virtualiseur utilisé est configurable dans les paramètres du programme, dans l'onglet Compiler.

6 EXPORTATION D'UN RAPPORT

JasperReports offre la possibilité d'exporter les rapports dans plusieurs formats de fichier :

- PDF
- XLS (Excel)
- RTF
- ODF
- HTML
- XML
- CSV
- Texte brut

L'exportation se réalise par l'intermédiaire de classes étendant l'interface `JRExporter` de JasperReports. Les deux méthodes principales de cette interface sont :

- `void setParameter (JRExporterParameter parameter, Object value)` : pour renseigner les paramètres nécessaires à l'exportation
- `void exportReport()` : pour lancer l'exportation du rapport

Dans la plupart des cas, deux paramètres doivent être renseignés :

- l'instance d'objet de classe `JasperPrint`, représentant le rapport rempli au format natif
- le flux ou fichier de sortie utilisé pour l'export

La classe `JRExporterParameter` contient un certain nombre de constantes représentant les paramètres pouvant être passés à l'exportateur, dont les paramètres ci-dessus via les constantes :

- `JRExporterParameter.JASPER_PRINT` pour l'instance d'objet de classe `JasperPrint`
- `JRExporterParameter.OUTPUT_FILE_NAME` pour le nom du fichier de sortie
- `JRExporterParameter.OUPUT_STREAM` pour le flux de sortie

L'exportation d'un rapport suit toujours la même logique. Voici quelques exemples d'exportation :

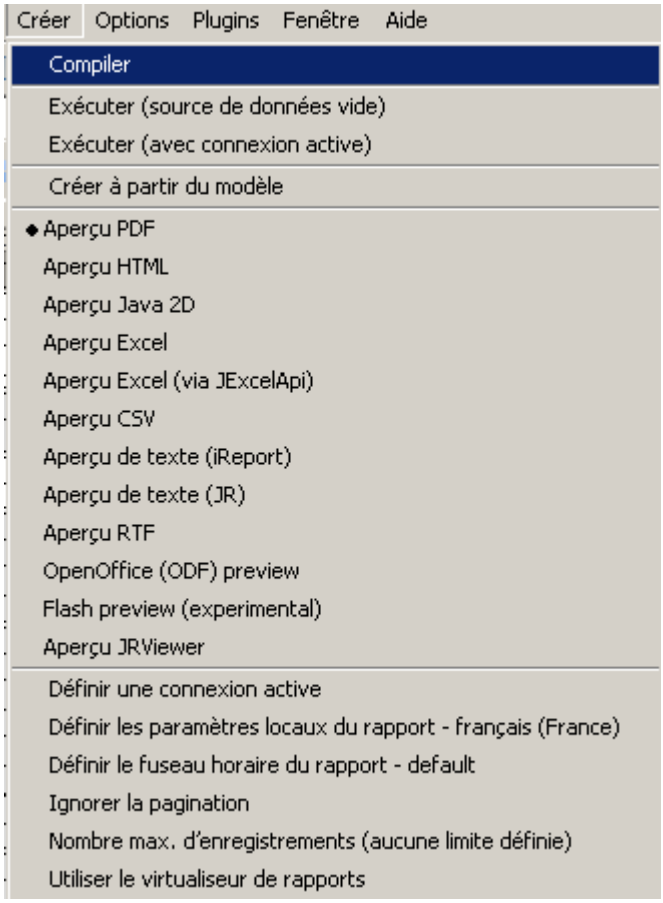
```
HashMap<String, Object> params = new HashMap<String, Object>();
// ...
// Renseignment des paramètres
// ...

//Remplissage du rapport
JasperPrint jasperPrint = JasperFillManager.fillReport(
                "rapport.jasper", params,
                new JREmptyDataSource());

// export en pdf
JRPdfExporter pdfExporter = new JRPdfExporter();
pdfExporter.setParameter(JRPdfExporterParameter.JASPER_PRINT,
                jasperPrint);
pdfExporter.setParameter(JRPdfExporterParameter.OUTPUT_FILE,
                new File("rapport.pdf"));
pdfExporter.exportReport();
```

Génération en PDF

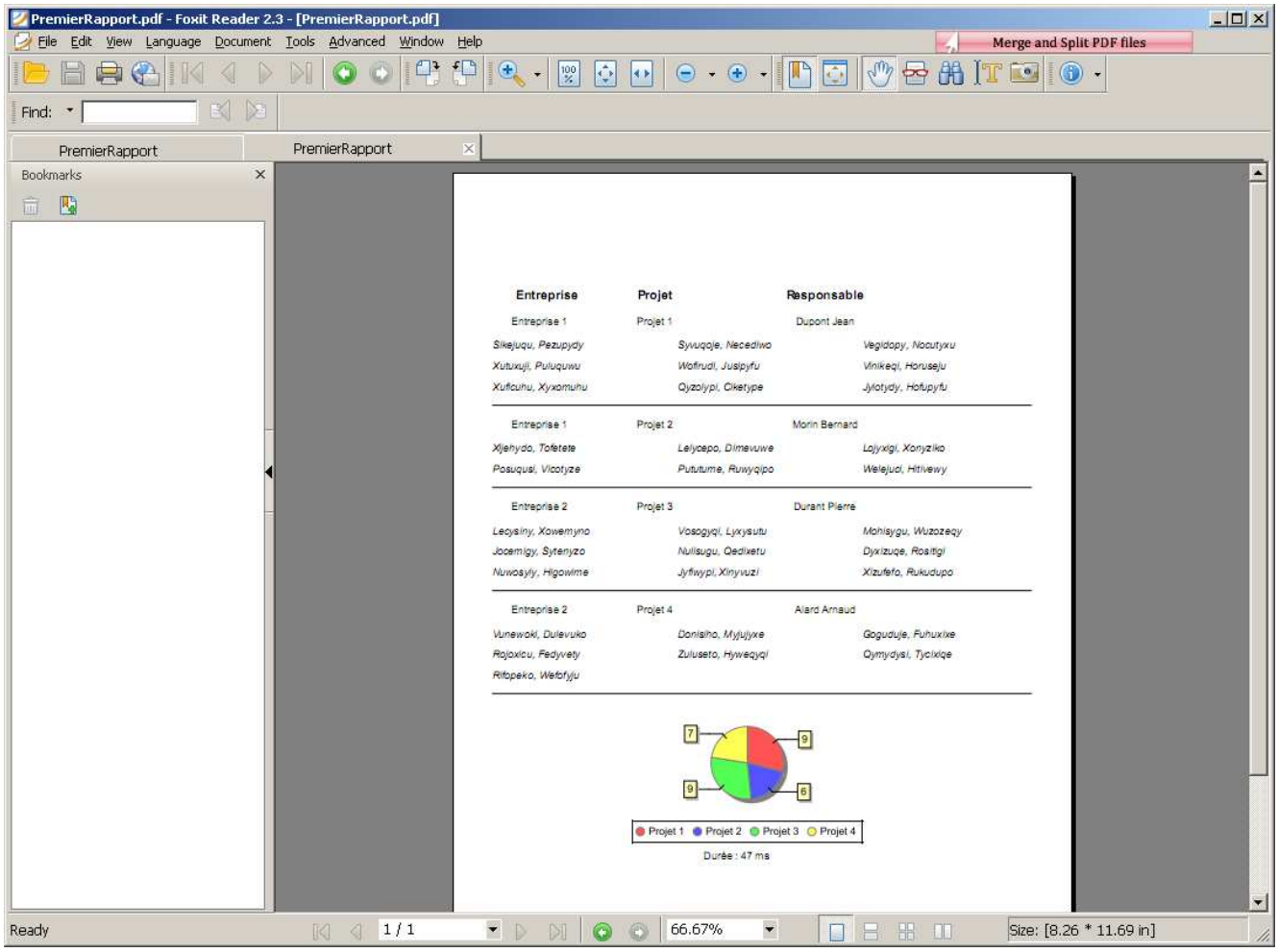
```
HashMap<String, Object> params = new HashMap<String, Object>();
// ...
// Renseignment des paramètres
// ...

//Remplissage du rapport
JasperPrint jasperPrint = JasperFillManager.fillReport(
                "rapport.jasper", params,
                new JREmptyDataSource());

// export en xls
JRXlsExporter excelExporter = new JRXlsExporter();
excelExporter.setParameter(JRPdfExporterParameter.JASPER_PRINT,
                jasperPrint);
excelExporter.setParameter(JRPdfExporterParameter.OUTPUT_FILE,
                new File("rapport.xls"));
excelExporter.exportReport();
```

Génération en XLS

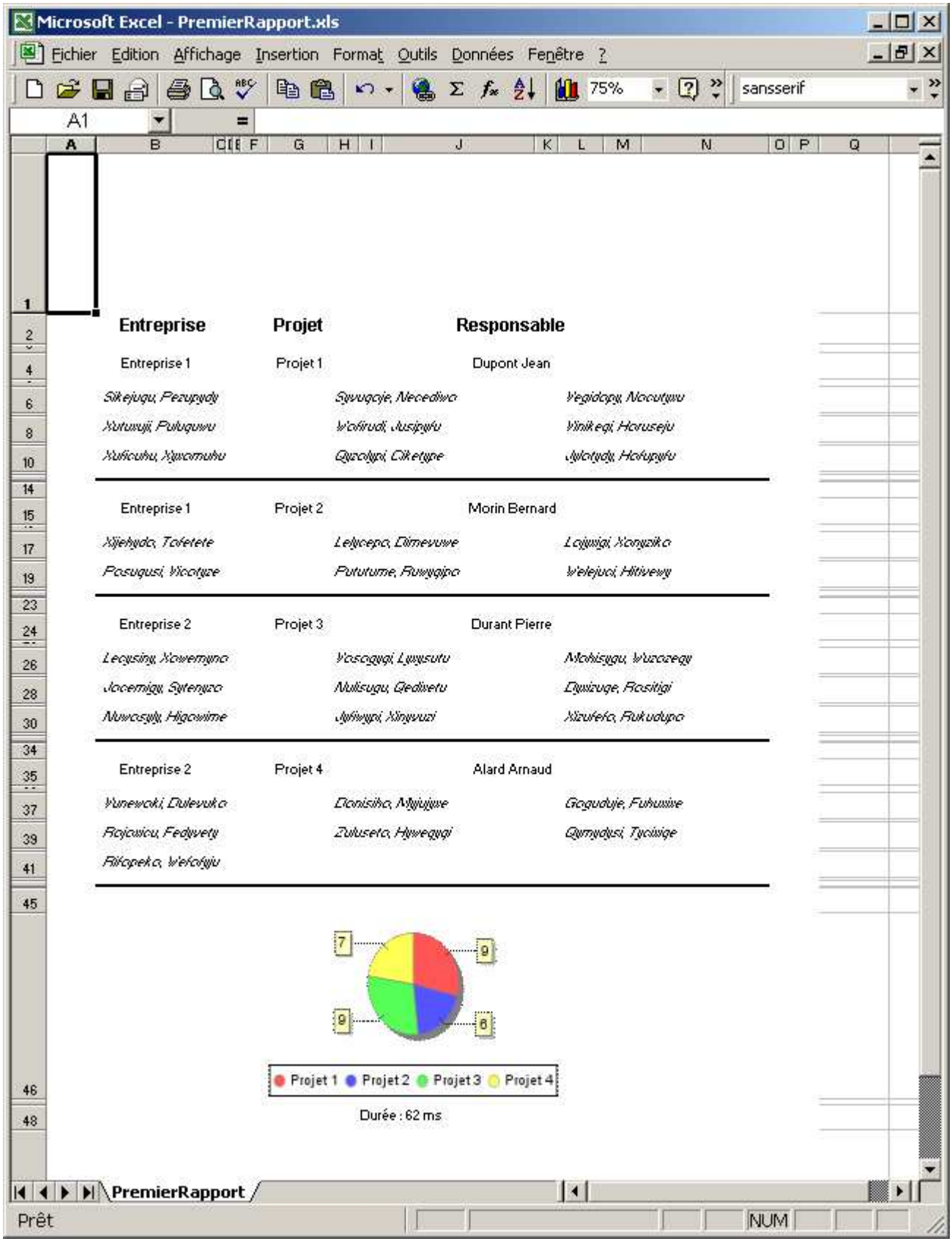
Dans iReport, le Menu Créer propose la génération dans les formats supportés par JasperReports :



En reprenant notre exemple du chapitre précédent :



Génération en PDF



Génération en XLS