

# Geração de Relatório com JasperReports e iReport

**Carlos Feliz Paixão**

*Este documento descreve a instalação, configuração e utilização de duas ferramentas open-source para a geração de relatórios em Java: **JasperReports** e **iReport**. É realizado um exemplo de geração de relatório no formato PDF, utilizando uma base de dados em Firebird, demonstrando assim o poder dessas ferramentas.*

## Introdução

Dentre as tarefas de um sistema, a mais comum é a geração de relatório. Presente na maioria dos sistemas, mas muitas vezes não suficientemente reconhecida, esta tarefa constitui um importante módulo do sistema.

Basicamente, o processo de geração de relatório resume-se na definição do design e mapeamento de dados para campos dentro de um layout definido. Nesse contexto, surgiram ferramentas comerciais com intuito de auxiliar neste processo. No passado, essa área foi completamente dominada por produtos comerciais, como o Crystal Reports, que com o passar do tempo tornaram-se cada vez mais robustos no que diz respeito a novas funcionalidades, como o suporte a diferentes tipos de fontes de dados. Porém, o que se vê hoje é o surgimento de ferramentas open-source com o mesmo objetivo, e tão ou mais robustas que as comerciais, com a grande conveniência de serem gratuitas. É o caso das ferramentas foco deste documento: JasperReports e iReport..

### Link's:

JasperReports: <http://jasperreports.sourceforge.net>  
iReport: <http://ireport.sourceforge.net>

## JasperReports

JasperReports é um poderoso framework open-source para geração de relatórios. Escrito em Java, essa biblioteca apresenta grande habilidade na organização e apresentação de conteúdo, permitindo a geração dinâmica de relatórios em diversos formatos, como PDF, HTML, XLS, CSV e XML, podendo ainda ser utilizada em qualquer aplicação Java, incluindo aplicações desktop, Web e distribuídas.

### Funcionamento

Antes de iniciar a utilizar a biblioteca JasperReports, é necessário a compreensão de seu funcionamento (Veja a fig. 1).

O design do relatório, incluindo a localização dos campos a serem preenchidos e seus respectivos nomes, para futuro mapeamento, são definidos em um arquivo **XML** através de tags XML que obedecem a uma estrutura, vocabulário e restrições declarados em um arquivo DTD (`jasperreports.dtd`). Usando XML, o designer pode definir textos estáticos, imagens, linhas, formas geométricas, como retângulos e elipses, e suas localizações dentro do relatório. Pode-se, ainda, e principalmente, definir os campos que serão preenchidos dinamicamente a partir de uma base de dados. O arquivo XML precisa ser compilado, gerando um arquivo **.jasper**, isto é, contendo a versão compilada do código XML. Isto implica na compilação de todas as expressões Java definidas no arquivo XML, sendo realizadas várias verificações em tempo de compilação.

Diferentes objetos JasperReports são usados para representar as etapas do processo de geração de relatório:

**JasperDesign:** Representa a definição do relatório. A partir do *template* XML é criado um JasperDesign.

**JasperReport:** Representa o JasperDesign compilado. O processo de compilação verifica o design do relatório e compila o design em um objeto JasperReport.

**JasperPrint:** Representa o relatório gerado. É criado um JasperPrint a partir de um JasperReport, contendo o relatório preenchido.

### Dados

Para produzir um relatório precisamos fornecer dados ao Jasper. Estes dados podem ser recuperados de diferentes lugares, como de uma base de dados em um SGBD ou em um arquivo XML. Para recuperarmos informações de um banco de dados relacional, precisamos realizar uma consulta (*query*)

em linguagem SQL. Essa query pode ser inserida ao código XML ou ser realizada por uma classe Java, gerando um objeto `ResultSet`, que será passado às classes do Jasper para o preenchimento do relatório.

O JasperReports suporta vários tipos de **datasources** (fonte de dados) através de uma interface específica chamada **JRDataSource**. Há uma implementação padrão desta interface para objetos `ResultSet`, chamada **JRResultSetDataSource**, ou seja, é possível realizar consultas, gerando objetos `ResultSet` e passando ao JasperReports para o preenchimento do relatório. Quando a fonte de dados é um `ResultSet`, este objeto deve conter todas as colunas a serem mapeadas para seus campos correspondentes no relatório.

A figura 1 ilustra o funcionamento do JasperReports:

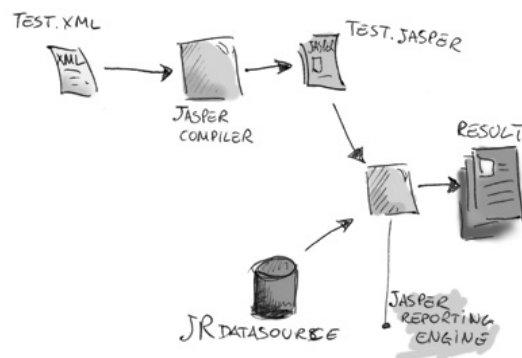


Fig. 1: Etapas para a geração de relatório com JasperReports.

No linguajar "Jasper", um **datasource** somado a um arquivo **.jasper** gera um "print", que pode ser "exportado" para os formatos PDF, HTML, XML, CVS ou XLS.

## Campos, Parâmetros, Variáveis e Expressões

**Campos (Fields)** são "áreas específicas" que receberão diretamente os dados das respectivas colunas referenciadas. O relatório deve conter um campo com o mesmo nome da coluna a qual faz referência. Por exemplo, para os dados da coluna **Nome** do tipo `VARCHAR`, da tabela **Cliente**, serem mapeados para o relatório, um campo **Nome** deve ser definido no arquivo XML da seguinte forma:

```
<field name="Nome" class="java.lang.String"/>
```

**Parâmetros** são dados passados para a operação de preenchimento, que não podem ser encontrados normalmente na fonte de dados. São declarados, por exemplo, da seguinte forma:

```
<parameter name="TituloDoRelatorio" class="java.lang.String"/>
```

E passados via código Java, através da classe `HashMap`:

```
Map parametros = new HashMap( );  
parametros.put( "Cliente", "Carlos Paixão" );
```

Outra importante utilização de parâmetros é na query do relatório. Por exemplo:

```
select * FROM CLIENTE WHERE CLIENTE=$P{Cliente}
```

O relatório será gerado apenas para o cliente passado por parâmetro.

**Variáveis** são utilizadas para simplificar o projeto do relatório. Através de uma variável podemos definir somente uma vez uma **expressão**, que seja usada frequentemente durante o design do relatório, chamando-a quando precisarmos daquela funcionalidade. Elas podem referenciar tipos internos de cálculos, como contagem (count), soma (sum), média (average), menor (lowest), maior (highest), etc. Por exemplo, o cálculo do valor total da compra:

```
<variable name="ValorTotalCompraSum" class="java.lang.Double" calculation="Sum">  
    <variable expression> ${ValorProduto} </variable expression>  
</variable>
```

Em uma expressão, uma variável pode referenciar outras variáveis do relatório, mas somente se aquelas variáveis foram definidas previamente no projeto do relatório. Assim, a ordem em que as variáveis são declaradas no relatório é importante.

Para as variáveis que executam o cálculo nós podemos especificar o nível em que devem ser reinicializadas. O nível **Report** (de relatório) significa que a variável será inicializada somente uma vez, no começo do relatório, e que executa o cálculo especificado até que o fim do relatório seja alcançado. Mas nós podemos escolher executar o cálculo **em nível de página, coluna** ou de **grupo**. O exemplo abaixo demonstra o mesmo cálculo anterior em nível de página. Nossa variável será inicializada com zero no começo de cada nova página:

```
<variable name="ValorTotalCompraSum" class="java.lang.Double"
resetType="Page" calculation="Sum">
    <variable expression> ${ValorProduto} </variable expression>
    <initialValueExpression> new Double( 0 ) </initialValueExpression>
</variable>
```

Existem também variáveis internas da ferramenta, com nomes "auto-explicativos", prontas para o uso nas expressões: PAGE\_NUMBER, COLUMN\_NUMBER, REPORT\_COUNT, PAGE\_COUNT, COLUMN\_COUNT, GroupName\_COUNT.

**Expressões (Expressions)** são utilizadas para especificar o conteúdo de campos de texto, na realização de cálculos freqüentes, por exemplo. Todas elas são expressões Java que podem conter em sua sintaxe: campos, parâmetros e variáveis de relatório. Por exemplo:

```
<textFieldExpression>
    "Sr.(a) " + $F{Cliente} + " realizou um total de compras no valor de "
    + $V{ValorTotalCompraSum} + " no dia "
    + (new SimpleDateFormat("dd/MM/yyyy")).format($F{DataCompra}) + "."
</textFieldExpression>
```

## Layout

Para a melhor organização e definição do design do relatório, o JasperReports divide o layout em áreas "pré-definidas", chamadas seções. Ao projetar um relatório nós precisamos definir a posição do conteúdo dentro de uma seção, levando em consideração o que ela representa na estrutura visual de um relatório. A seguir estão as seções em que é baseado o layout de relatório: **title, pageHeader, columnHeader, groupHeader, detail, groupFooter, columnFoter, pageFooter, summary**.

### Mais detalhes

Enquanto escrevia, o JasperReports encontrava-se em sua versão 0.5.x, na qual se baseia este documento. Maiores informações em:  
<http://jasperreports.sourceforge.net>.

## iReport

Criar o design do relatório diretamente em XML pode ser uma tarefa custosa. Necessitava-se, então, de uma ferramenta que automatizasse esse processo. O iReport veio preencher essa lacuna, permitindo definir o design do relatório dentro de um ambiente gráfico, contento "todos" os recursos que a biblioteca Jasper oferece. É possível definir relatórios com designs modernos e complexos sem se quer escrever uma linha de código XML, que é todo gerado automaticamente. O ambiente ainda oferece atalhos para tarefas de compilação e visualização do relatório, permitindo a realização de testes, acelerando assim o processo de design.

É importante salientar que existem outras ferramentas com o mesmo objetivo que o iReport, mas que não são suficientemente maduras, no que diz respeito a facilidade de uso, e principalmente, no suporte as tags XML do JasperReports.

### Mais detalhes

Abaixo estão outras alternativas para o design gráfico do relatório:

- JasperDesign (<http://sourceforge.net/projects/jasperdesign>)
- JasperEdit (<http://sourceforge.net/projects/jasperedit>)
- OpenReports (<http://sourceforge.net/projects/oreports>)
- JasperJEdit (<http://www.sourceillustrated.com/jasperjedit>)
- JasperPal (<http://sourceforge.net/projects/jasperpal>)

## Fonte de Dados

O iReport dá suporte a conexões JDBC, ODBC, e à 4 tipos de datasources:

**1 – Empty data source (JREmptyDataSource):** é um especial datasource usado para preencher relatórios que não possuem registros ou dados recuperados. Este datasource é usado quando é pressionado o botão “run”:



**2 – XML DataSource:** é um datasource capaz de empacotar um arquivo XML e normalizar seu conteúdo. As únicas informações necessárias para criar este tipo de datasource são: o nome do datasource e o nome do arquivo XML.

**3 – JavaBeans Set DataSource:** é um datasource capaz de empacotar uma Collection ou um Array de JavaBeans. É necessário uma classe especial de fábrica (factory) que forneça um método estático para gerar a coleção ou um array de JavaBeans. Para criar este datasource você precisa de um nome para o datasource, o nome da classe que fornece o método estático para recuperar o Array/Collection de objetos e o nome deste método, que terá uma definição como esta:

```
public static Collection createMyCollection( )
```

ou

```
public static Object[ ] createMyArray( )
```

É necessário setar o tipo de resultado (Collection ou Array).

**4 – Custom DataSource:** este tipo de datasource é genérico. O iReport não sabe como a interface JRDataSource é implementada por esta conexão particular, mas isto não é importante. É necessário uma classe especial de fábrica (factory) que forneça um método estático que retorne um JRDataSource. Para criar este datasource você precisa do nome do datasource, do nome da classe que fornece o método estático para recuperar o JRDataSource, e do nome deste método que terá uma definição como esta:

```
public static JRDataSource createMyJRDataSource( )
```

### Mais detalhes

Enquanto escrevia, o iReport encontrava-se em sua versão 0.2.x, na qual se baseia este documento. Maiores informações em: <http://ireport.sourceforge.net>.

## Requisitos

Ambas as ferramentas possuem requisitos para o seu funcionamento, os quais são em sua maioria bibliotecas de terceiros, também gratuitas. Ao realizar o download opte pelos arquivos completos, pois estes possuem todas as bibliotecas necessárias. Não é necessário baixar o jasperreports, pois o iReport já o traz em sua pasta lib, entretanto, é interessante ver os exemplos e o javadoc fornecidos pelo JasperReports.

### Mais detalhes

Você pode ver a lista completa de requisitos para o JasperReports e iReport nos link's abaixo:

JasperReports: <http://jasperreports.sourceforge.net/requirements.html>

iReport: <http://ireport.sourceforge.net/cap2.html#2.1>

## Instalando e configurando

**1.** Descompacte os arquivos do iReport no diretório de sua preferência. Cheque nesse momento se serão criadas as pastas referentes.

**2.** Cheque se o JDK e o Ant estão devidamente configurados, ou seja, se as variáveis de ambiente **JAVA\_HOME** e **ANT\_HOME** foram criadas e estão apontando para os diretórios corretos. Lembre-se de setar o **PATH** com **JAVA\_HOME\bin** e **ANT\_HOME\bin**, e o **CLASSPATH** com **JAVA\_HOME\lib** e **ANT\_HOME\lib**. No caso dos dados a serem usados forem recuperados de um banco de dados você deve também setar no CLASSPATH o driver JDBC do banco.

**3.** Copie o arquivo **tools.jar**, localizado na pasta **lib** de seu JDK instalado, para os diretório **...\iReport\lib**

**4.** Copie o driver JDBC de seu banco para a pasta “lib” do iReport.

**5.** Para testar a instalação do iReport, abra o prompt, vá ao diretório do iReport e digite **ant iReport**. Outra opção é editar o arquivo **iReport.bat** (para Windows) ou **iReport.sh** (para Linux), localizados na

pasta iReport-0.2.x. Corrija as variáveis para apontarem para os diretórios corretos, executando o arquivo de lote. Se tudo ocorreu bem o iReport irá inicializar. Há também uma forma de iniciar o iReport **sem ter o Ant instalado**. Para isso, execute o arquivo de lote chamado **startup.bat** (Windows) e **statrtup.sh** (Linux) localizados na pasta noAnt do diretório principal do iReport.

6. Ao iniciar, vá ao menu **Tools >> Options**, irá surgir uma janela (vide Fig. 2) onde você deve localizar a aba **External Programs**, setando ali todos os programas externos a serem utilizados como visualizadores de relatório. Após isso clique em **Save**.

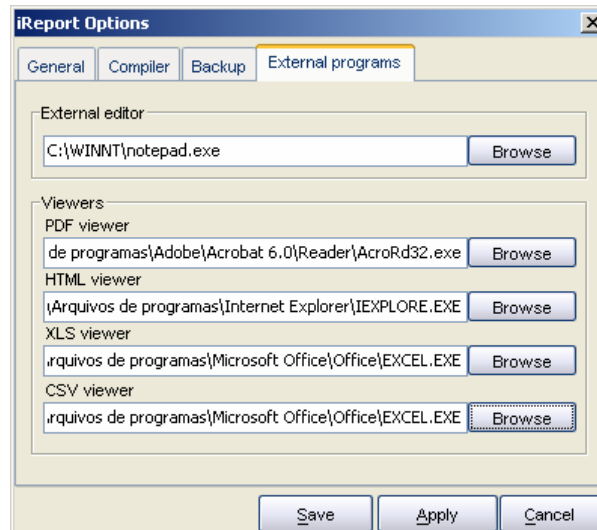


Fig. 2: Setando programas externos.

## Exemplo

O exemplo a seguir utiliza como fonte de dados uma base simples no SGBD open source Firebird, mas você pode adaptar o exemplo para uma base e um SGBD qualquer, desde que este possua um driver JDBC ou ODBC, devidamente configurado. O driver JDBC tipo 4 (conexão direta com o banco) do Firebird é chamado de Jaybird.

### Mais detalhes

Abaixo estão os link's para realizar o download dos arquivos e ferramentas necessárias para o exemplo:

Firebird - <http://firebird.sourceforge.net>

Jaybird - [http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp\\_download](http://www.ibphoenix.com/main.nfs?a=ibphoenix&page=ibp_download)

Base de dados – [acervo.fdb](http://www.acervo.fdb)

**Obs.:** Este documento não descreve como utilizar outros tipos de datasources, isto será feito em outra oportunidade.

### Criando uma conexão

Antes de mais nada devemos criar uma conexão com nossa fonte de dados:

1. Selecione o menu **Datasources >> Connection/Datasources**. A janela que aparece lista todas as fontes de dados e conexões JDBC que você define. Clique no botão **New** para criar uma nova conexão (ou datasource).

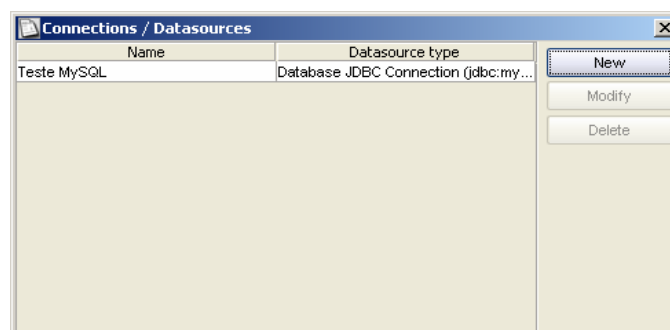


Fig. 3: Lista de conexões / datasources.

2. Na janela que aparece escolha o tipo de conexão (no nosso caso Database JDBC Connection). Na mesma janela dê um nome à conexão. Por exemplo, **"BibliotecaPessoal FB"**.

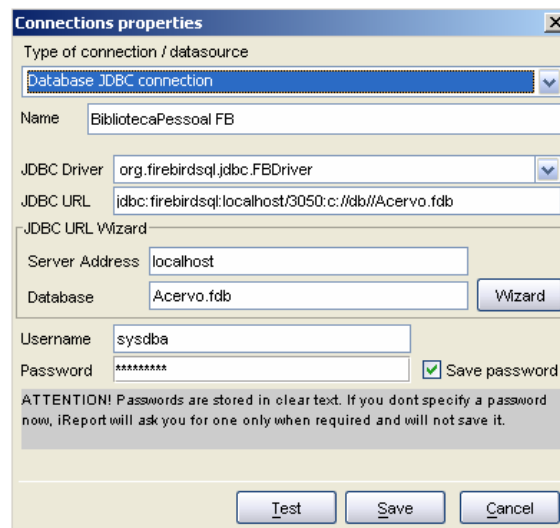
3. No campo **"JDBC Driver"**, selecione o driver do seu banco. Caso não esteja listado, como no nosso caso, especifique um de acordo com o seu SGBD. Para o Firebird, digite **"org.firebirdsql.jdbc.FBDriver"**, lembrando que o driver deve estar no classpath.

4. Em **"JDBC URL"** digitamos o caminho para a nossa base de dados:

**"jdbc:firebirdsql:localhost/3050:c://db//Acervo.fdb"**

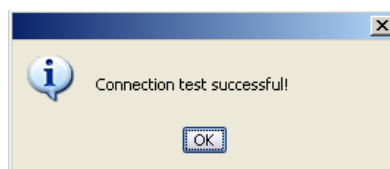
**Obs.:** A sintaxe da URL difere de SGBD para SGBD, consulte a documentação de seu banco para saber a sintaxe exata. Repare também no caminho para a base de dados.

5. Em **"Server Address"** digite o caminho para servidor de banco de dados, no nosso caso **"localhost"**. Em **"Database"**, entre com o nome do banco (**Acervo.fdb**) ou do *alias*. Entramos, então, com o user **"sysdba"** e password **"masterkey"**. Temos a possibilidade de salvar a senha marcando a opção **"Save password"**.



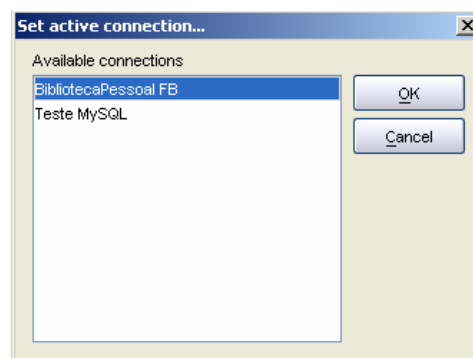
**Fig. 4:** Criando uma nova conexão JDBC.

6. Para testarmos a conexão clicamos no botão **"Test"**. Não esqueça que o SGBD precisa estar rodando. Se tudo ocorreu bem a seguinte mensagem irá aparecer:



**Fig. 5:** Teste de conexão bem sucedido.

7. O próximo passo é ir ao menu **"Build >> Set active connection"**, escolhendo na janela que aparece a conexão com a base que iremos utilizar para preencher o relatório, no nosso caso: **"BibliotecaPessoal FB"**.



**Fig. 6:** Escolhendo a conexão.

**Obs.:** É importante salientar que a configuração de uma conexão dentro do iReport não é obrigatória, pois podemos utilizá-lo apenas para criar o design do relatório e compilá-lo. As tarefas de preenchimento, exportação para um formato (pdf, por exemplo), e visualização, ficam de responsabilidade de uma classe

Java. Porém, é conveniente, até mesmo para a realização dos testes, termos um único ambiente que além de permitir a definição do layout, também permita visualizarmos, em diversos formatos, o resultado do design e preenchimento do relatório através de um simples botão (Run): . Para isso, é necessário que a conexão JDBC esteja funcionando.



## Criando o Design do Relatório

Considerando que o teste de conexão com o banco foi bem-sucedido iniciamos a etapa de design do relatório.

1. Vá ao menu **"File >> New Document"** ou clique no botão:



2. Irá aparecer a janela **"Report Properties"** onde damos um nome ao nosso relatório (**Report Name**) chamando-o de **"RelatorioBibliotecaPessoal"** (não pode haver espaço entre as palavras). Ainda na mesma janela podemos configurar o tamanho da página, a sua orientação, as margens, o comprimento, largura e o espaço entre as colunas, além de outras opções mais avançadas.

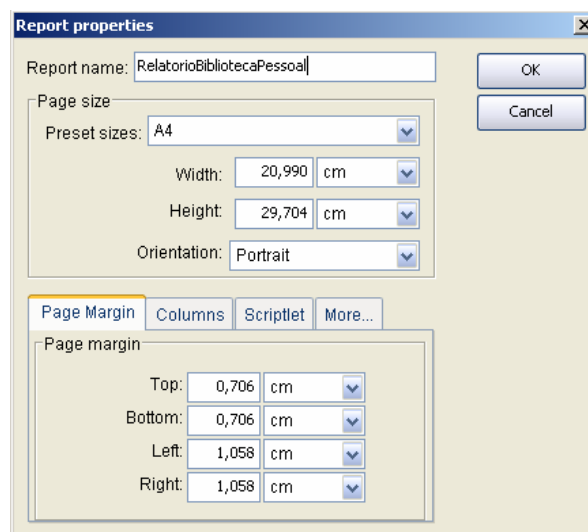




Fig. 7: Configurando a página do relatório.

3. Agora podemos criar o design de nosso relatório (fig. 8), utilizando a barra de ferramentas:



Inserimos os elementos do relatório, bastando clicarmos uma vez sobre o respectivo botão e logo após no relatório. O conteúdo dos elementos de texto e as configurações de fonte estão disponíveis através de um duplo clique sobre o elemento, assim como as configurações para os elementos geométricos.



Na figura 8, os textos **"Relatório Biblioteca Pessoal"**, **"Data:"**, **"Código"**, **"Título"**, **"Volume"**, **"Edição"**, **"Editora"**, **"Autor"** e **"Adquirido em"** são campos de texto estático: 

A tabela é constituída de dois Retângulos "arredondados": , enquanto os outros elementos são todos campos (fields).

|                              |             |            |             |             |               |                     |
|------------------------------|-------------|------------|-------------|-------------|---------------|---------------------|
| Relatório Biblioteca Pessoal |             |            |             |             |               |                     |
| Data: new                    |             |            |             |             |               |                     |
| pageHeader                   |             |            |             |             |               |                     |
| Código                       | Título      | Autor      | Edição      | Volume      | Editora       | Adquirido em        |
| \$F                          | \$F{TITULO} | \$F{AUTOR} | \$F{EDICAO} | \$F{VOLUME} | \$F{EDITORIA} | \$F{DATA_ADQUIRIDO} |
| columnFooter                 |             |            |             |             |               |                     |
| pageFooter                   |             |            |             |             |               |                     |
| "Página " + \$V              |             |            |             |             | \$V           |                     |
| summary                      |             |            |             |             |               |                     |

Fig. 8: Design do relatório.



4. Para definir os campos (fields) para preenchimento , precisamos conhecer os nomes das colunas das tabelas de onde recuperaremos os dados. O iReport permite a inserção do SQL no código XML através do menu **"Datasource >> Report query"** ou clicando no botão: 

5. No campo da janela que aparece, na aba **Report SQL Query**, digitamos nossa query. Levando em consideração que o resultado dessa consulta deve conter os dados que preencherão o relatório, o SQL deve ser bem estruturado. A mesma janela nos permite visualizar os nomes dos campos que fazem parte do resultado da consulta. Por exemplo, se digitarmos a consulta **SELECT \* FROM LIVRO** e em seguida clicarmos no botão **"Read Fields"** visualizaremos todas as colunas da tabela LIVRO de nosso banco, e seus respectivos tipos. Para inserir esta consulta no código XML devemos clicar no botão **"Save query to report"**.

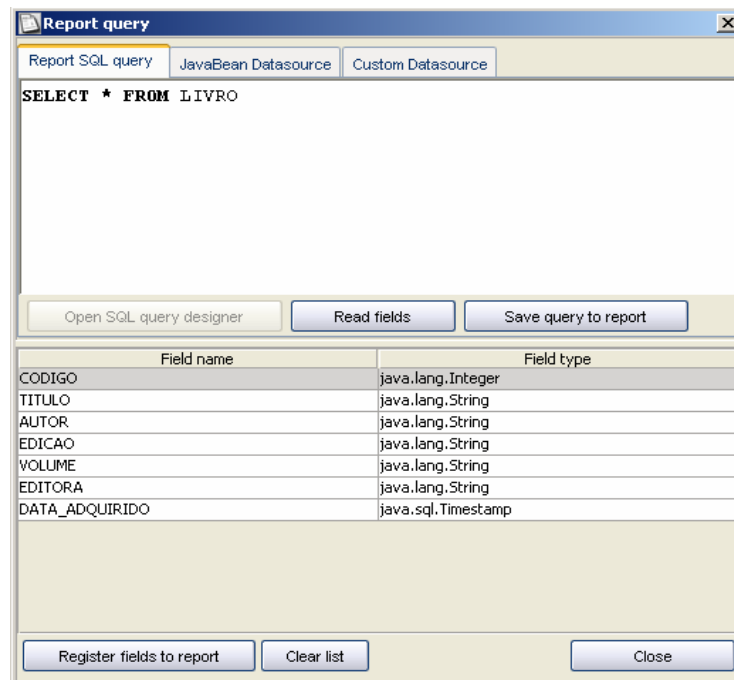



Fig. 9: Visualizando os nomes das colunas contidas no resultado da consulta.

**Obs.:** Em nosso exemplo não iremos inserir o SQL no código XML. Realizaremos a conexão com o banco e a consulta em nossa classe Java, passando o objeto ResultSet gerado para o JasperReports.

6. Agora que conhecemos os nomes das colunas e seus tipos, podemos definir nossos campos (fields) no relatório, **registrando-os** para o mapeamento dos dados. Para tanto, na mesma janela selecione o primeiro campo (coluna) que aparece e, segurando a tecla Shift, clique no último campo, pressionando o botão **"Register fields to report"**. A mesma tarefa pode ser realizada indo ao menu **"View >> Report fields"** ou clicando no botão: 

A janela que aparece permite o registro de campos, parâmetros e a criação de variáveis no relatório:

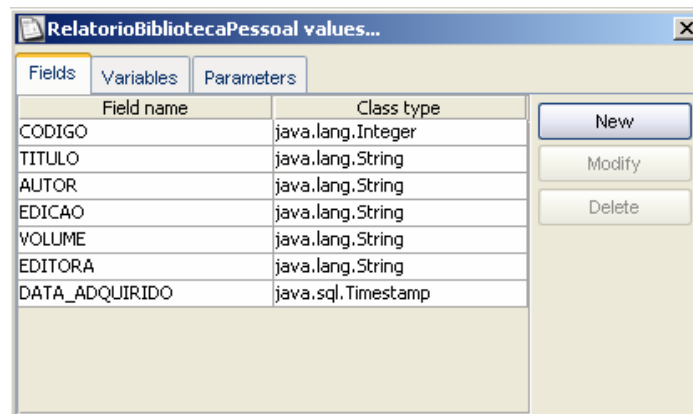


Fig. 10: Registrando os campos (fields).



7. Após registrarmos as colunas a serem mapeadas, devemos inserir elementos fields que farão referência a elas para serem preenchidos. Para isso, depois de inserir um "field", damos um duplo clique nele, surgindo assim uma janela onde, na aba "Text Field", em "TextField Expression Class", configuraremos a classe referente ao tipo da coluna que queremos mapear; e em "Textfield expression", o nome desta coluna entre **\$F{ e }**. Por exemplo, para mapear a coluna CODIGO que é do tipo INTEGER, selecionamos em "TextField Expression Class", a classe **java.lang.Integer**, e em "Textfield expression" digitamos **\$F{CODIGO}**. Realize o mesmo processo para as outras colunas (TITULO, AUTOR, VOLUME, EDICAO, EDITORA, DATA\_ADQUIRIDO), atentando para o tipo da coluna.

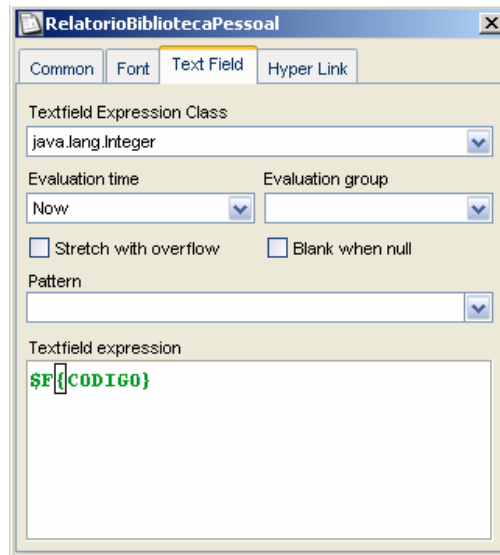


Fig. 11: Mapeando a coluna CODIGO.

8. Podemos também criar expressões para efetuar cálculos ou formatações sobre os dados provenientes das colunas mapeadas, assim como, optar a que nível executar esta expressão. (veja **Variáveis e Expressões**)

Em nosso exemplo inserimos a data atual de "impressão" do relatório, formatando-a de acordo com o padrão brasileiro. Para isso, inserimos um "field", damos um duplo clique sobre ele, e na janela que aparece, na aba "Text Field", digitamos a seguinte expressão Java no campo "Textfield Expression":

```
new SimpleDateFormat( "dd/MM/yyyy" ).format( new Date( ) )
```

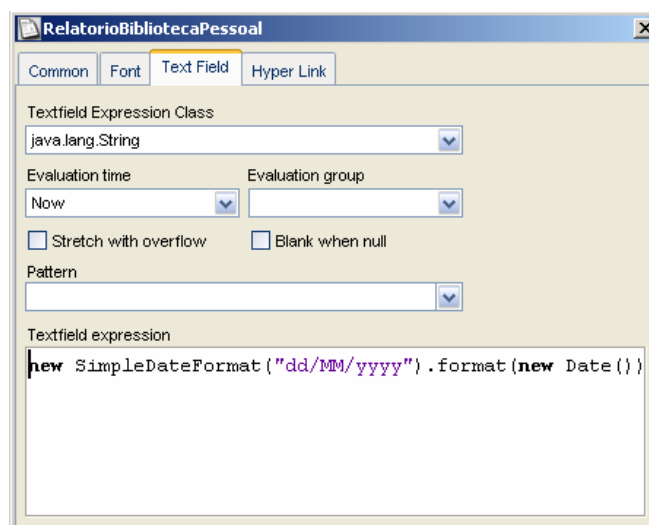


Fig. 12: Inserindo a data atual formatada no relatório.

Outro exemplo pode ser visto ao inserirmos o número das páginas de nosso relatório. Para isso, utilizamos a variável interna **\$V{PAGE\_NUMBER}**.

Inserimos dois “fields” no relatório. No primeiro, no campo “Evaluation time”, escolhemos o nível de execução “**Now**”, que considerará a página atual e para o segundo field, o nível “**Report**”, que significa que a variável será inicializada somente uma vez, no começo do relatório, e que executa o cálculo especificado até que o fim do relatório seja alcançado, ou seja, a quantidade de total de páginas.

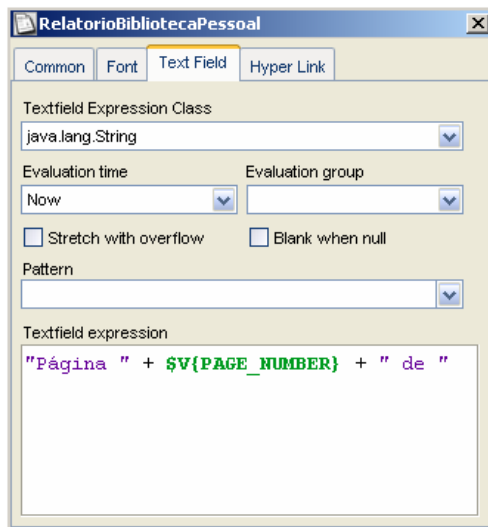


Fig. 13: Número da página atual.

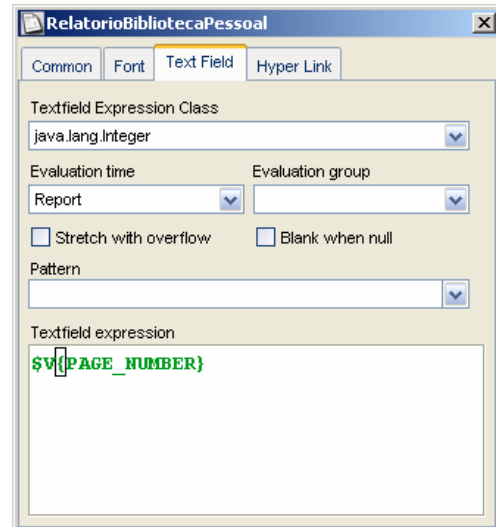


Fig. 14: Número total de páginas.

9. Para informarmos ao iReport em que formato desejamos visualizar o relatório, vamos ao menu “**Build**”, escolhendo uma das opções de visualização ali existentes.

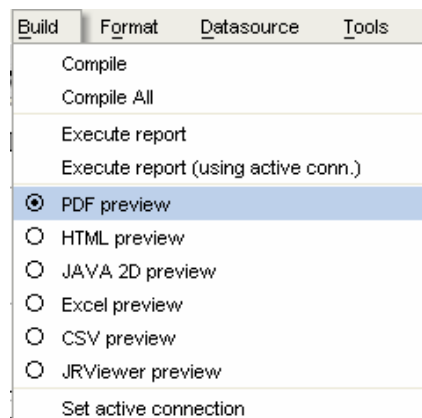



Fig. 15: Opções de visualização.

Obs.: Podemos agora realizar um teste, clicando no botão .

## Gerando relatório a partir de uma classe

Vimos até o momento como configurar e utilizar o iReport para montar o layout de nosso relatório, e também visualizá-lo para fins de teste. Nesta seção, veremos na prática como gerar um relatório em formato pdf a partir de uma classe simples. A classe `RelatorioBibliotecaPessoal` será apresentada e descrita posteriormente.

Supõe-se que você já utilizou o iReport para criar o design do relatório e definir os campos a serem preenchidos, que estas informações foram salvas em um arquivo XML, e que este foi compilado neste mesmo ambiente, gerando um arquivo `.jasper`. Este último arquivo (`.jasper`) será utilizado pela classe a seguir:

```
/* pacotes necessários */
import dori.jasper.engine.*;
import dori.jasper.view.JasperViewer;
import java.sql.*;
import java.util.*;
```

```
public class RelatorioBibliotecaPessoal {

    /* Realiza conexão com o banco de dados, retornando o objeto Connection */
    private static Connection getConnection( ) throws
ClassNotFoundException, SQLException
    {
        String driver = "org.firebirdsql.jdbc.FBDriver";
        String url = "jdbc:firebirdsql:localhost/3050:c://db//Acervo.FDB";
        String user = "sysdba";
        String password = "masterkey";

        Class.forName(driver);
        Connection con = DriverManager.getConnection( url, user, password );
        return con;
    }

    /* Gera Relatorio e visualiza-o */
    public void geraRelatorio( ) throws JRException, Exception
    {
        Connection con = getConnection( );
        Statement stm = con.createStatement( );
        String query = "select * from Livro";
        ResultSet rs = stm.executeQuery( query );

        /* implementação da interface JRDataSource para DataSource ResultSet */
        JRResultSetDataSource jrRS = new JRResultSetDataSource( rs );

        /* HashMap de parametros utilizados no relatório. Sempre instanciados */
        Map parameters = new HashMap();
        // parameters.put("COLUNA", valor);

        /* Preenche o relatório com os dados. Gera o arquivo BibliotecaPessoal.jrprint */
        JasperFillManager.fillReportToFile( "BibliotecaPessoal.jasper", parameters, jrRS );

        /* Exporta para o formato PDF */
        JasperExportManager.exportReportToPdfFile( "BibliotecaPessoal.jrprint" );

        /* Preenche o relatorio e o salva diretamente em arquivo PDF. Sem
        a necessidade do .jrprint */
        // JasperRunManager.runReportToPdfFile("BibliotecaPessoal.jasper", parameters, jrRS);

        /* Visualiza o relatório em formato PDF */
        JasperViewer.viewReport( "BibliotecaPessoal.pdf", false );
    }

    public static void main(String[] args) throws JRException, Exception
    {
        new RelatorioBibliotecaPessoal().geraRelatorio();
    }
}
```

Basicamente, a classe anterior realiza os seguintes passos:

1. O método `geraRelatorio()` chama o método `getConnection()` para realizar a conexão com o banco de dados e retornar um objeto `Connection`.
2. É realizada a consulta sql, gerando um objeto `ResultSet` com os dados referentes ao preenchimento do relatório. Este objeto é passado ao construtor da classe `JRResultSetDataSource` (implementação padrão da interface `JRDataSource` para `ResultSet`) do JasperReports, criando o objeto `datasource` (`jrRS`).
3. O objeto `datasource` (`jrRS`) é passado como parâmetro para o método estático `fillReportToFile()` da classe `dori.jasper.engine.JasperFillManager` para o preenchimento do relatório, assim como o XML compilado (`BibliotecaPessoal.jasper`) e possíveis parâmetros (`parameters`). Este procedimento cria o arquivo (`BibliotecaPessoal.jrprint`) referente ao relatório preenchido.

**Obs.:** O HashMap de parâmetros deve ser instanciado, mesmo não havendo a utilização destes no relatório. São exigidos por todos os métodos de preenchimento do JasperReports.

4. O arquivo `.jrprint` (`BibliotecaPessoal.jrprint`) é passado como parâmetro para o método `exportReportToPdfFile()` da classe `dori.jasper.engine.JasperExportManager` para exportar o relatório para o formato pdf.

#### Mais detalhes

Outros formatos suportados pelo JasperReports podem ser obtidos a partir da utilização de seus métodos referentes. Consulte a documentação (javadoc) do JasperReports para conhecê-los: <http://jasperreports.sourceforge.net/api/index.html>

**Obs.:** A classe `dori.jasper.engine.JasperRunManager` pode substituir a utilização das classes `JasperFillManager` e `JasperExportManager` nos passos 3 e 4. Seu método `runReportToPdfFile()`, por exemplo, recebe como parâmetros o arquivo `.jasper` referente ao relatório compilado, possíveis parâmetros (`parameters`) e o objeto `JRResultSetDataSource`.

5. Finalizando, é utilizada a classe `dori.jasper.viewer.JasperViewer`, uma GUI para a visualização do relatório em pdf. Seu método `viewReport()` recebe como parâmetro o nome do arquivo de relatório e um boolean referente ao formato do arquivo de relatório, ou seja, `true` para arquivos XML e `false` para outro.

#### Conclusão

Demonstramos a interação do JasperReports com o iReport, para a geração de relatórios. Vimos como instalar e configurar o iReport, assim como utilizá-lo para desenhar relatórios estruturados que serão processados pelo JasperReports.

Ambas as ferramentas demonstradas aqui, se completam e mostram-se poderosas, oferecendo ao desenvolvedor um amplo conjunto de recursos na organização e apresentação de conteúdo de um relatório. A utilização destas, permite o rápido desenvolvimento de um relatório estruturado e complexo de modo ágil e facilitado.

**Carlos Feliz Paixão** ([carlosfpaixao@yahoo.com.br](mailto:carlosfpaixao@yahoo.com.br)) é programador Java e estagiário de desenvolvimento do Serviço Federal de Processamento de Dados (Serpro) – Departamento SUNAT - Regional Belém-PA - usando JasperReports ([jasperreports.sourceforge.net](http://jasperreports.sourceforge.net)) e iReport ([ireport.sourceforge.net](http://ireport.sourceforge.net)) para a geração de relatórios.