SpagoBI QuickStart

Authors

Luca Scarel
Grazia Cazzin
Davide Zerbetto
Luca Fiscato

Index

Version

Version/Release n°:	0.6	**Data Version/Release:**	December,14th 2005
Update description:	First release – Draft		
Version/Release n°:	0.7	**Data Version/Release:**	January, 31st 2006
Update description:	Second release – Draft		
Version/Release n°:	0.8	**Data Version/Release:**	April, 11th 2006
Update description:	Third release – Draft (Added paragraph on datamart jar creation)		
Version/Release n°:	0.9	**Data Version/Release:**	July, 20th 2006
Update description:	Fourth release – Draft		
Version/Release n°:	0.9.1	**Data Version/Release:**	October, 9th 2006
Update description:	Fifth release - Draft		
Version/Release n°:	0.9.2	**Data Version/Release:**	January, 24nd 2007
Update description:	Sixth release - Draft		

1. Document Goal

The document aim is to introduce the reader to the SpagoBI concepts by means of a full example based on the "SpagoBI demo distribution" . The demo is freely downloading from the ObjectWeb forge (http://forge.objectweb.org/project/showfiles.php?group_id=204)

The document includes the following main chapters :

•**Conceptual overview**. Introduction of the core concepts of the SpagoBI free open source platform.

•**Functionality overview**. An explanation of the SpagoBI portlets and of the main tasks of the various users typologies, considering the SpagoBI demo as an example.

•**Getting started with SpagoBI**. How to build step-by-step an analytic application by means of the case study available in the SpagoBI demo distribution. Starting from the development of a template, you are introduced to the document parametrization and configuration following a logical path. The chapter uses simple examples to explain the SpagoBI main concepts.

•**In more depth**. How to build an analytical portal in more detail

2. References

For further information about SpagoBI platform refer to the following documentation, available on the project site (http://spagobi.eng.it):

[1] Cazzin G., Ruffatti G., *SpagoBI Overview*

[2] Cazzin G., *SpagoBI Architectural Design*

[3] Zoppello A., SpagoBI Installation Manual

3. Help for lecture

Follows a short description of the most common views in SpagoBI.

3.1. Stylistic conventions

LITTLE CAPITALS	The LITTLE CAPITALS references to the icon in a mask.
italics	The *italics* refers to fields of the masks.
<ITALIC CAPITALS>	In *<ITALIC CAPITALS>* the logical variables are suitable.
boldface	In **boldface** the main concepts.

3.2. Special sections


Note


Example


Reference to other section


In revision phase


Future implementation. To be done.


Advice for the reading of the section

4. SpagoBI - Conceptual overview

SpagoBI is a platform for the development of Business Intelligence projects: SpagoBI offers all the tools and the necessary components for the realization of analytical portals, whose designing and setting are the main project activities.
The development of an analytical portal with SpagoBI doesn't require the implementation of some J2EE services; you have only to set up the analytical documents and to register them correctly in the platform.
The **analytical documents** provide the end user with the needed information, in the most suitable way. SpagoBI allows to use many categories of analytical tools: Report, OLAP, Data Mining, Dashboard, Visual Inquiry, Maps.

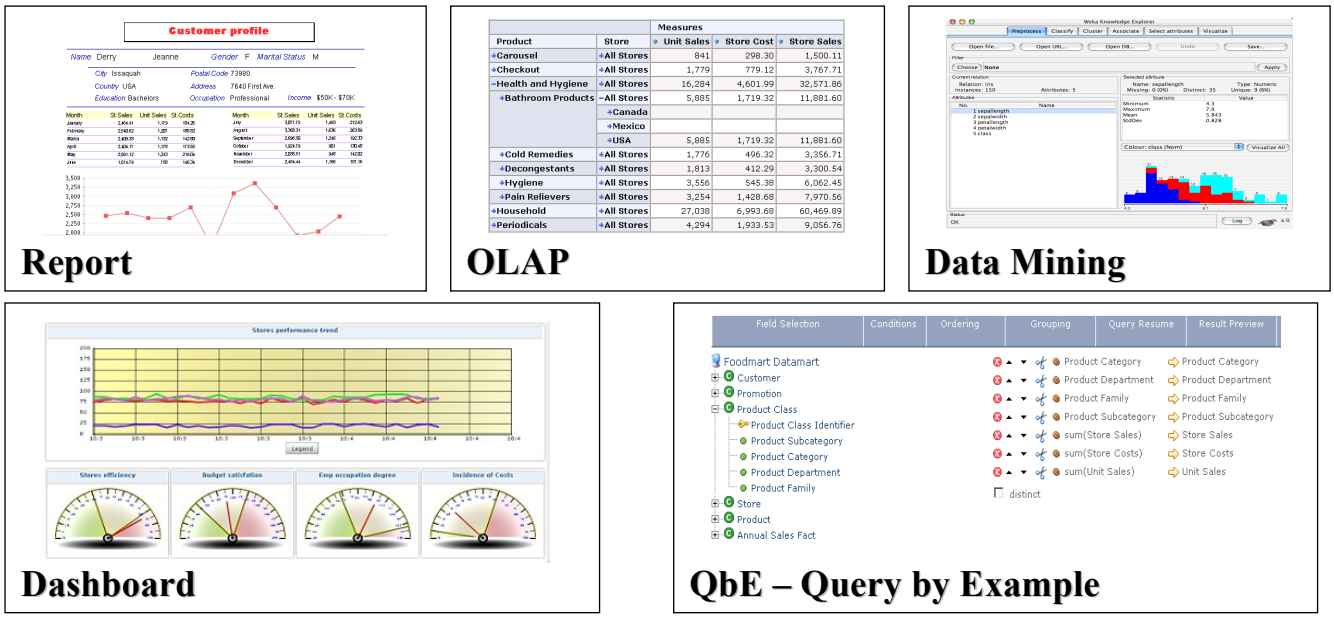

Figure 1 - Analytical documents

The Business Intelligence analyst and designer have to find the most suitable tool for every type of analysis and category of user. In fact, the building of an **analytical portal** is a balanced composition of different tools in order to give to each users' category the right degree of visibility and at the same time a freedom of movement through the information of his pertinence.
SpagoBI realizes both the structural and the executive support to the single analytical areas.

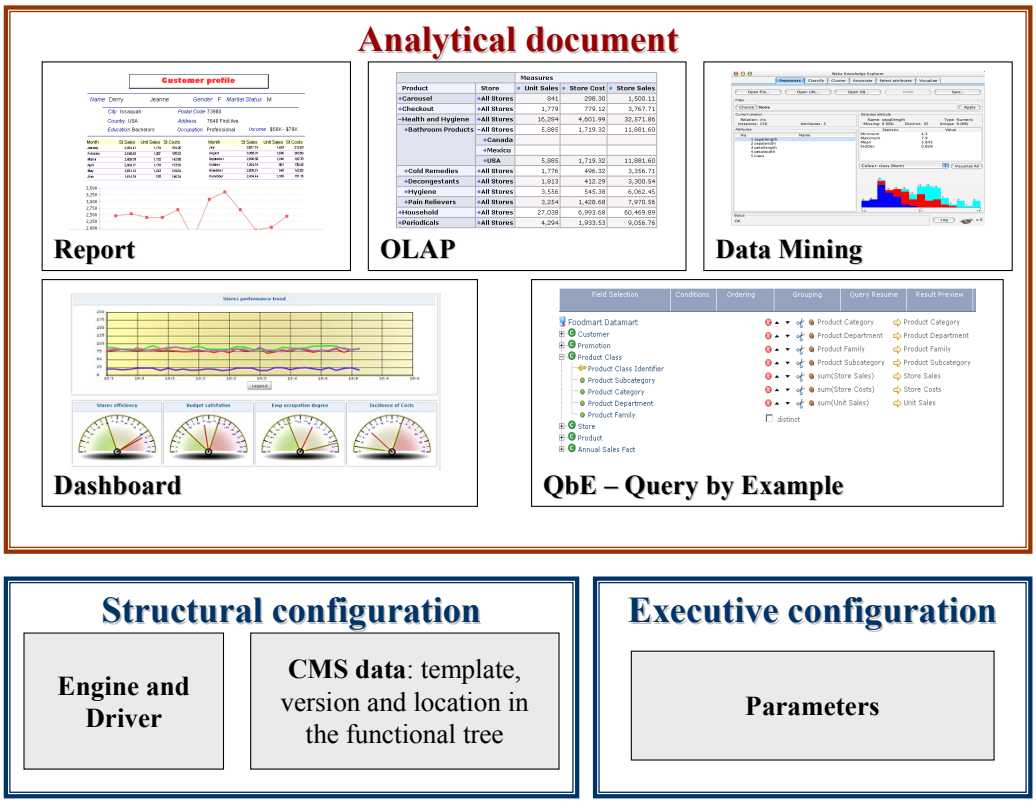

Figure 2 – Analytical documents support

Regarding to the **structural configuration**, every analytical documents' category refers to a particular (one or many) execution engine, interacting with SpagoBI through a specific driver. For every analytical document SpagoBI keeps the history of the templating version and organize them in the functional tree.
The SpagoBI demo has got a preloaded metadata environment allowing to manage:

- report on the JasperReport engine and Birt engine;
- OLAP on the Mondrian engine, with Jpivot interface;
- Dashboard on the OpenLazslo engine;
- free inquiry on a Hibernate implementation;
- data mining model on Weka engine.

The Dashboard implementation by means of OpenLaszlo compiler is under a phase of greater integration through.

Many other alternative engines in every analytical area will be integrated in the SpagoBI platform in the future.

Regarding to the **executive configuration**, SpagoBI manages some parameters as autonomous and independent entities. The parameters include the behaviour rules (presentation and validation) according to the end-user roles. Through the parameters,

SpagoBI builds an executive environment which places in the middle the operative model referred to the particular reality in use.

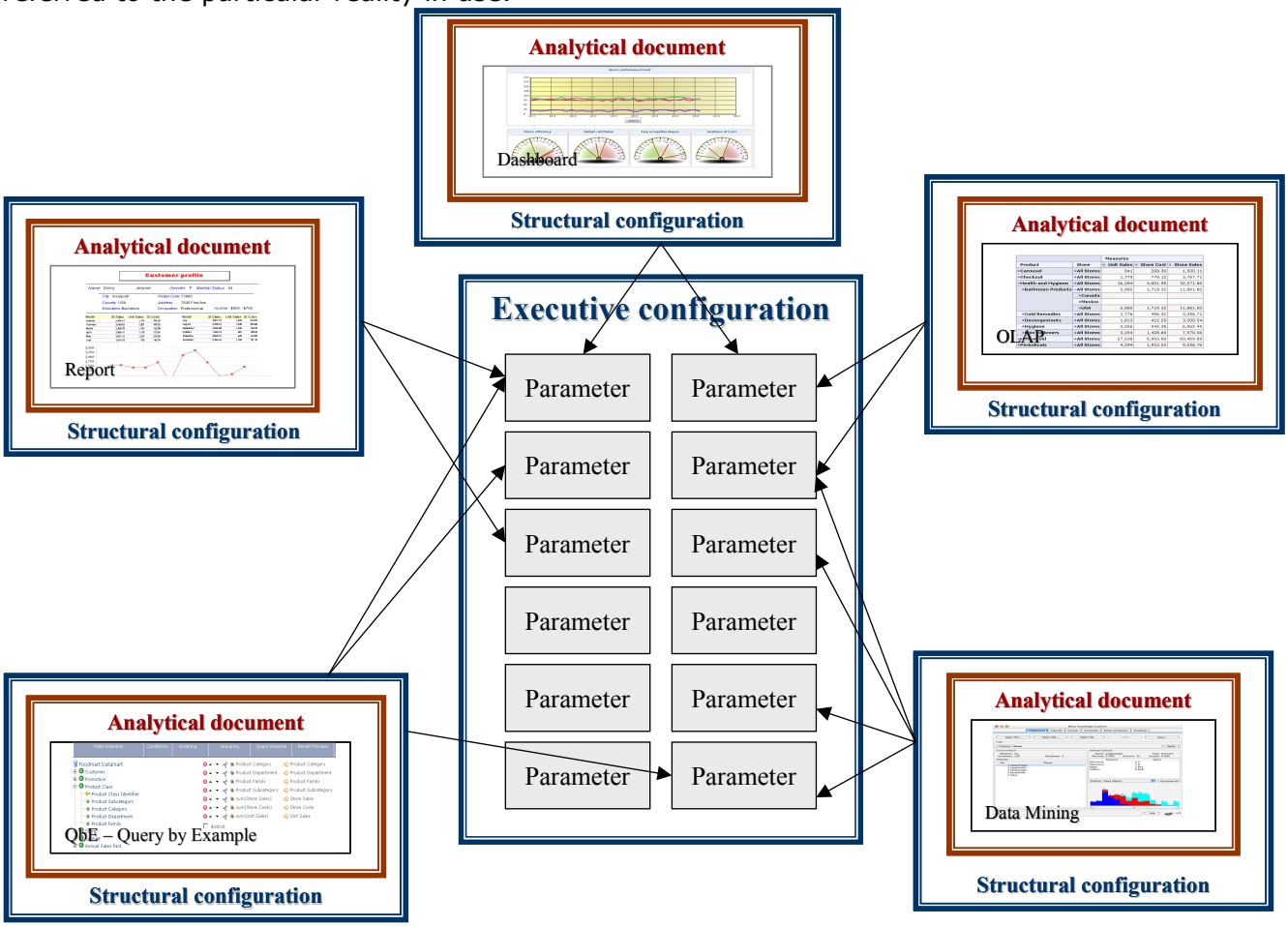

Figure 3 – Executive configuration

A **parameter** models a concept or a datum frequently used as discriminant on the global data context.
Every parameter can be used in many different ways, according to the different end-users' roles. So, every use mode refers to an initial visualization method, to one or more validation rules and to one or more end-user roles.
Follow a parameter abstract schema and an implementation example.

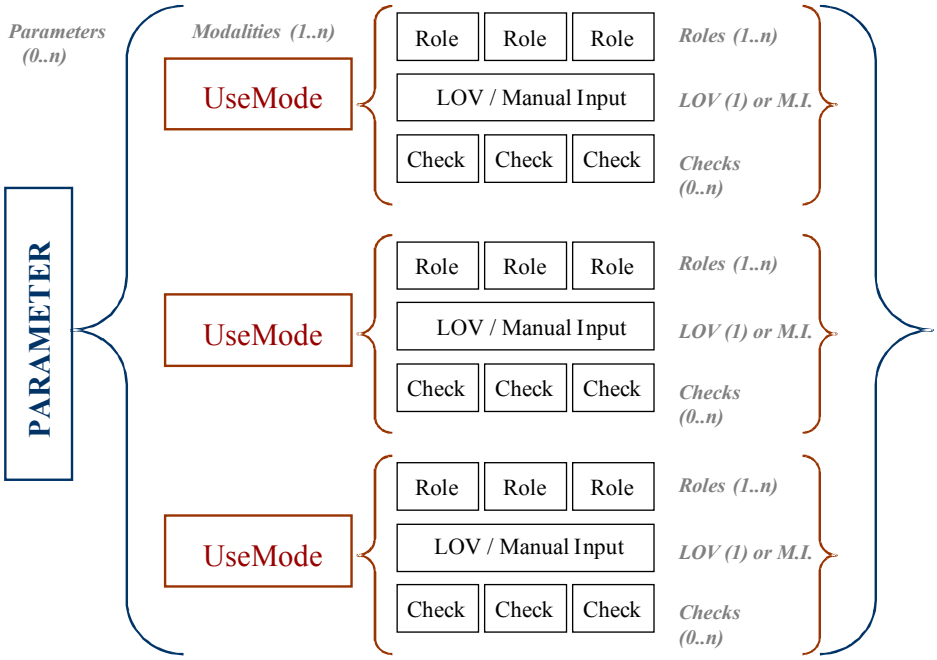

Figure 4 – Parameter's abstract structure

Figure 5 – Parameter's implementation example

Every parameter can be associated to many different analytical documents (also for category) driving their behaviour according to its rules.
The recording phase of an analytical document must therefore set its structural (driver, engine, CMS) and executive (parameters) information.
When a user (with its role) runs an analytical document, the structural information are read and then a custom page for the parametric input is produced on the basis of the execution information. At the end the document is produced on the basis of the inserted values (explicitly or implicitly).
A sample follows in the picture below:

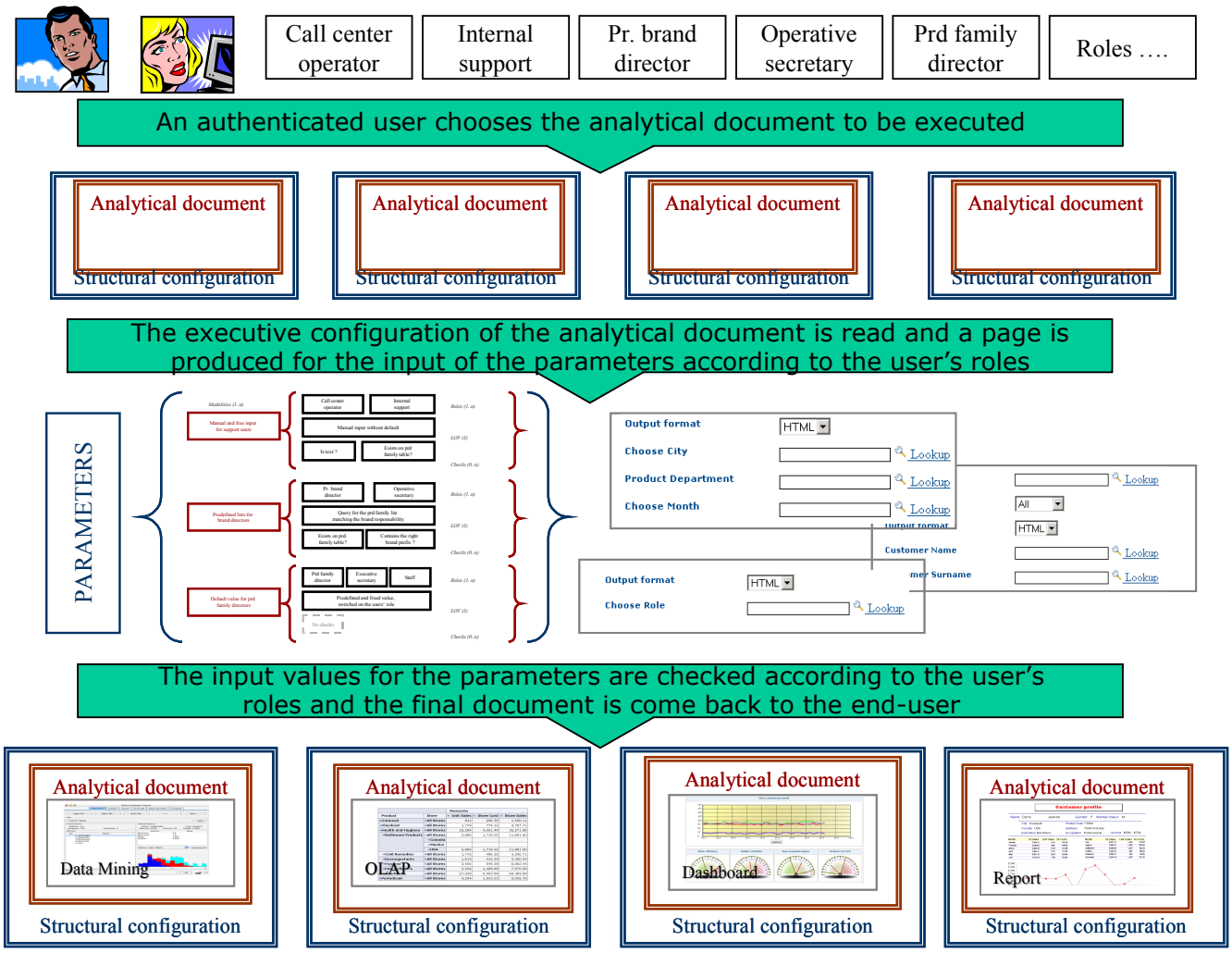

Figure 6 – Analytical document execution

A new analytical document requires a process handling as the one shown in the following schema:

.

Figure 7 – Analytical document handling

The schema points out also the management of the approval flow for the analytical document passing from the developer to the tester, reaching the end user when certified only.

Please see in the next paragraphs how SpagoBI allows to run all these operations.

5. Recurring themes

Both the development and the administration interface are under revision to achieve a better usability.

5.1. Portlet Layout

Every user portlet points out some common caractheristics:

- On the top, there is the title identifying the portlet meaning.
- On the right side of the title, some icons allow the access to the general functions acting on the portlet's content. The main functions are (where admitted):

• 	going back to the previous page without saving changes;	Every portlet
• 	creating a new element;	Every portlet
• 	switching from the list view to the tree view;	Document config.
• 	switching from the tree view to the list view;	Document config.
• 	saving information;	Details pages
• 	saving information and going back to the previous page;	Details pages
• 	testing before saving.	LOV details

- The '*' character identifies the required fields.

5.2. List and detailed View

One of the most common view in SpagoBI is a simple table showing a list of elements.

Common characteristics are:

- On the top, the title identifying the table meaning.
- The first row shows a label for each column displayed.
- The list can be divided into pages that can be turn over using the two arrows on the bottom row.
- The current page and the total number of pages are displayed in the middle of the bottom row.
- Every list has a detailed page showing and allowing to modify all the data about a single element.

Every list is alphabetically ordered on the first column's content (the label) and each row shows the essential data of an element, always identified by a unique label or title.

On the right side of every row, some icons drive the operativeness on the single element (row) of the list. The main possible functions are (where admitted):

• 	accessing the details page for the selected element (row).	Every list

- deleting the corresponding element (row); — Every list
- executing the corresponding element (row); — Analytical Doc. list only
- Selecting all. — Tree management

A standard view of a list and detailed page follows.

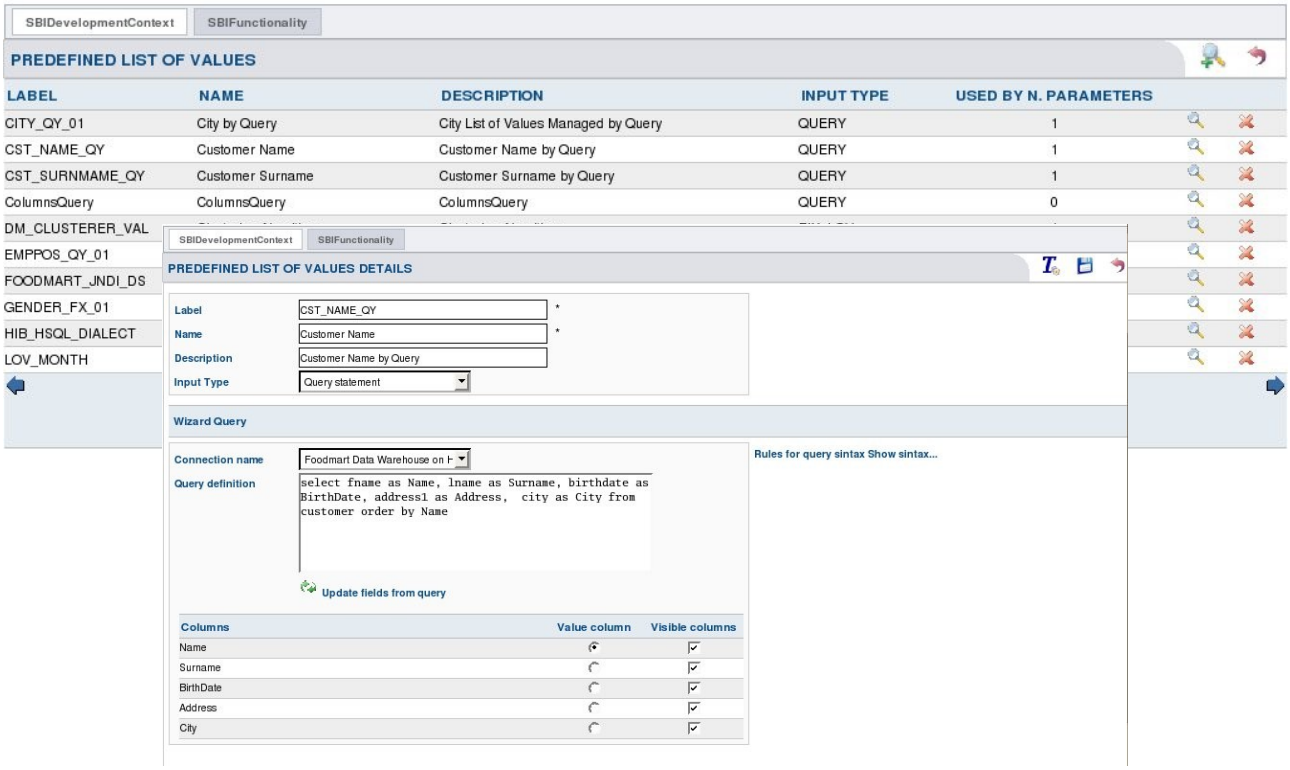

Figure 8 – List-detail Example

6. Functionality overview

6.1. Settings

In SpagoBI demo the administrator (biadmin/biadmin user) has some portlets; one of them is the SbiSettings portlet which permits him to:

- register and configure each analytical engine inside the platform;
- synchronize SpagoBI roles with the portal roles;
- import/export documents.

☺ If you do not have familiarity with SpagoBI yet, we suggest you to skip all this section (chapter 4.1) using the standard demo settings.

Figure 9 - Settings portlet

6.1.1. Engines Configuration

Engines are external applications or internal SpagoBI classes delegated to display the final results of an analytical document. There are different engines to deal with different analytical areas (Report, OLAP, Data Mining and Dashboard). More than one engine could be available for the same area.
The SpagoBI administrator has to mark the attributes that are necessary for the correct use of an engine. By means of a correct configuration the user can both use the same instances of the same engine inside different environments (development, test, production), in order not to invalidate its performances, and to use different and parallel engines inside the same environments.
For the correct use of the engines it is very important to set the proper **driver:** this is a SpagoBI component delegated to configure the analytical document properly communicating to a specific engine. Therefore, they can be seen as an Adapter set between the analytical document and the specific engine.

A list of all the registered engines can be displayed accessing the *Engine Configuration*.

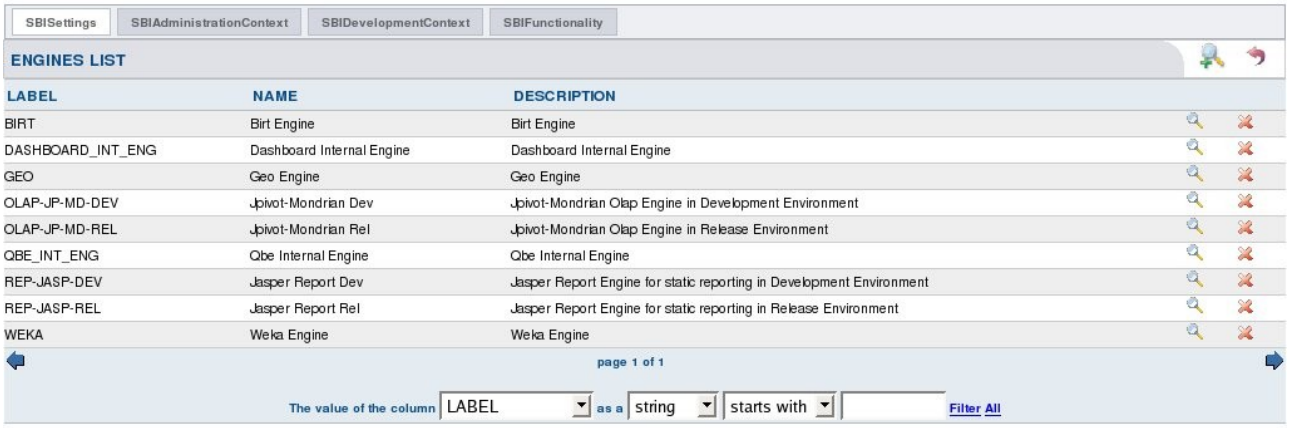

Figure 10 – Engines list

Each engine is depicted by a unique label, a name and a brief description. The administrator can create a new engine, erase an existing one or access their details page in order to change their configuration.

The information required for each engine are:

- **Label**: engine unique identifier;
- **Name**: engine name;
- **Description**: brief engine description (optional);
- **Document type**: the document type this engine will be applied to (Report/On-line analytical processing/Dashboard/Datamart Model/Data mining model/Map);
- **Engine type**: internal/external.

Engines can be internal or external (respect to the SpagoBI web application): in the former case you have to specify:

- **Class name**: name of the engine main class (it must implement the interface *it.eng.spagobi.engines.InternalEngineIFace*).

Figure 11 - Internal engine details

The current SpagoBI version uses the following internal engines:

- *Qbe Internal Engine*: for Datamart models;
- *Dahsboard Internal Engine*: for Dashboards.

It the latter case you have to specify:

- **URL**: location where the engine can be accessed by the server;
- **Driver Name:** class that creates an URL compliant with the specific engine.

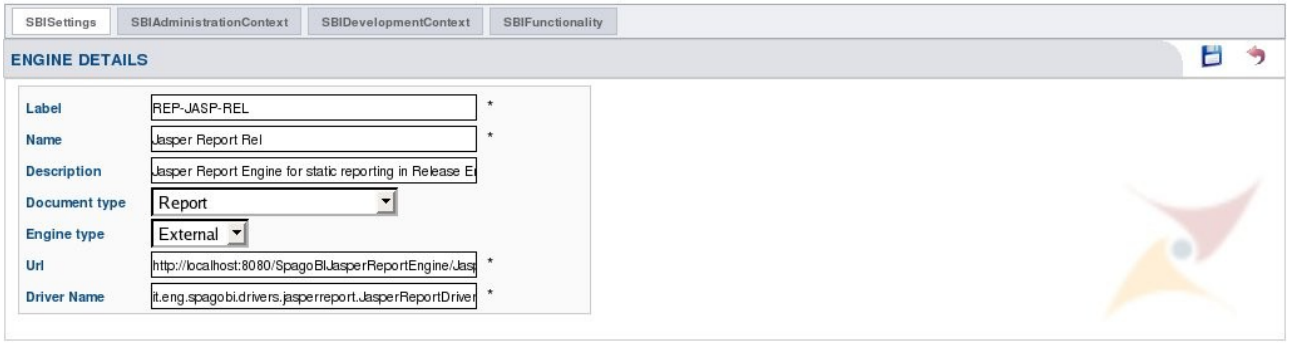

Figure 12 - External engine details

The current SpagoBI version uses the following external engines:

- *JasperReport*: for Report analysis;
- *Birt:* for Report analysis;
- *Jpivot*: for OLAP analyis;
- *Weka*: for Data mining models;
- *Geo*: for Maps.

6.1.2. Roles synchronization

A list of all SpagoBI's roles can be visualized by accessing to the *Roles Synchronization*. This area allows the administrator to synchronize portal roles with SpagoBI roles. Roles synchronization is made automatically by SpagoBI when the application starts, but it can be also forced by the administrator when the application has started.

Each role is described by two attributes:

- **Name**: the role's name;
- **Description**: the role's description.

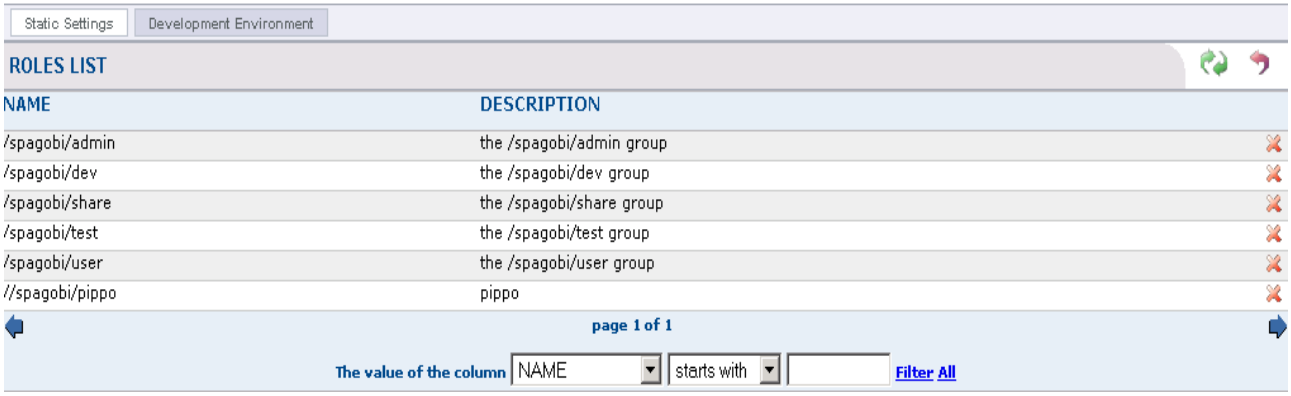

Figure 13 - Roles list

The administrator can:

- erase an existing role (if that role is not used by any functionality or parameter use)
- synchronize SpagoBI's roles with portal's roles by clicking on the green two-arrows button in the Roles List page.

The roles list will be reloaded according to the roles erasing or synchronization.

6.1.3. Import/Export

The import/export feature allows you to export one or more business objects from a SpagoBI platform and to import them into a different SpagoBI Installation. There can be many different reasons for an export operation but some typical cases can be backups, upgrade to a new platform version and deploy of business documents from a development environment to a production environment.

It's possible to access to the import/export feature from the SpagoBI configuration tools (settings portlet) selecting the 'Import/Export' link. Once entered into the Import / Export section you can see the main page with the export form on the left side and the import one on the right side.

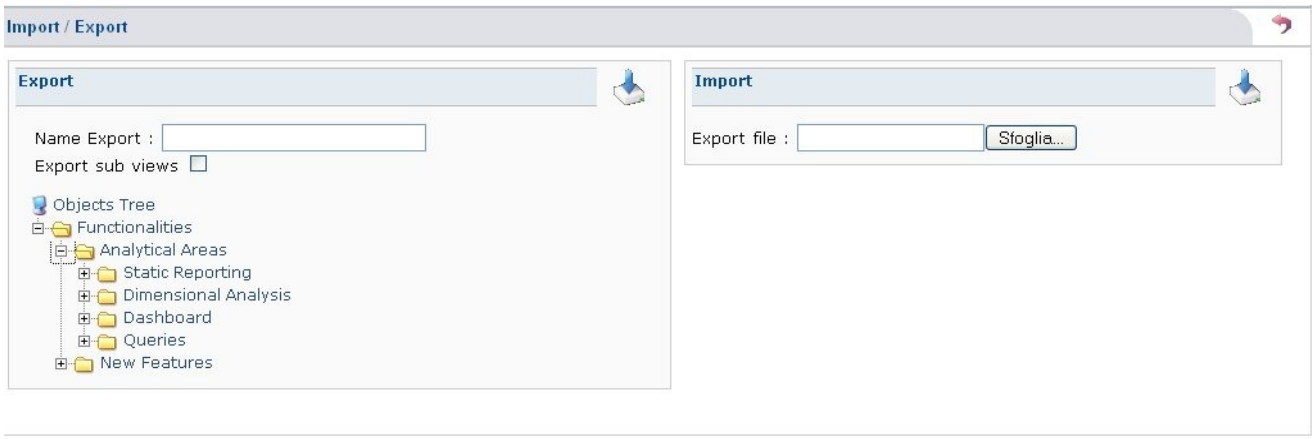

Figure 14 - Import / Export

6.1.4. Export

The export form allows you to:

- give a name to the export archive, that will be created, setting the 'Name Export' field; (mandatory)
- export the sub views of the exported objects, checking the 'Export sub views' option
- select the business objects to export, opening the tree and checking the checkbox corresponding to each document.

Into the tree you can select the documents one by one, or you can click over the folder row and, within the contextual menu, you can choose to select / deselect all the documents contained.

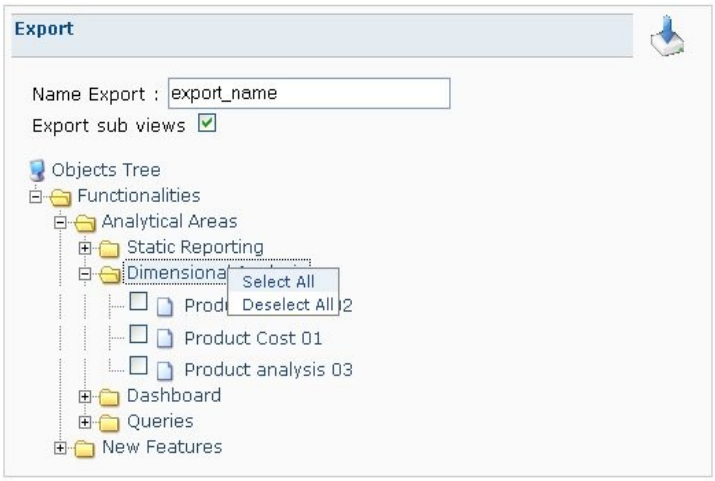

Figure 15 - Export documents selection

Once selected the documents to export, you can click the export image on the right upper corner. The system starts to produce the export archive and an 'operation in progress'

message will appear into the form. Once the archive production is terminated a 'download' link will appear. Click on it in order to download the export archive.

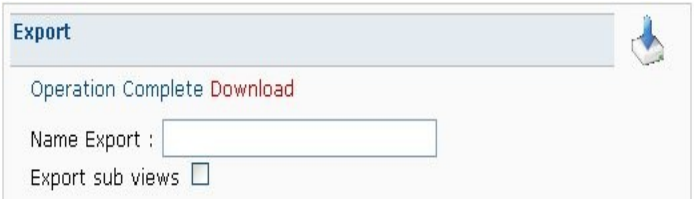

Figure 16 - Export operation complete

Once selected the download link you get a zip file, with name equals to the one setted into the export form, which contains all the information useful to import the documents into a different platform.

6.1.5. Import

The Import form is the start point of the import procedure. The 'Export File' selection field allows you to choose an exported archive from your file system.

Figure 17 - Import archive selection

Once selected the archive, you can click the import image on the right upper corner to start the import procedure. Since the the roles exported can be different form the ones of the current platform the next step require to map the roles.

Figure 18 - Import roles association

The form shows a the complete list of exported roles (roles associated to the exported documents) and, for each one of them, there's a combo which allows you to select the corresponding role into the import system. If a role with the same name of the exported one already exists, then the combo is disabled and the mapping is automatic. Otherwise, if the combo is selectable, you can choose the mapping role or you can let it blank, and, in this case, the exported role will be insert into the current system. Remember that each exported role can be associated only to one existing role.
Once complete the role mapping you can select the next button (the right arrows) which will allow you to perform the engine mapping. In fact, as for the roles, the exported engines should be different from the ones of the current system, and so, a mapping operation is necessary.

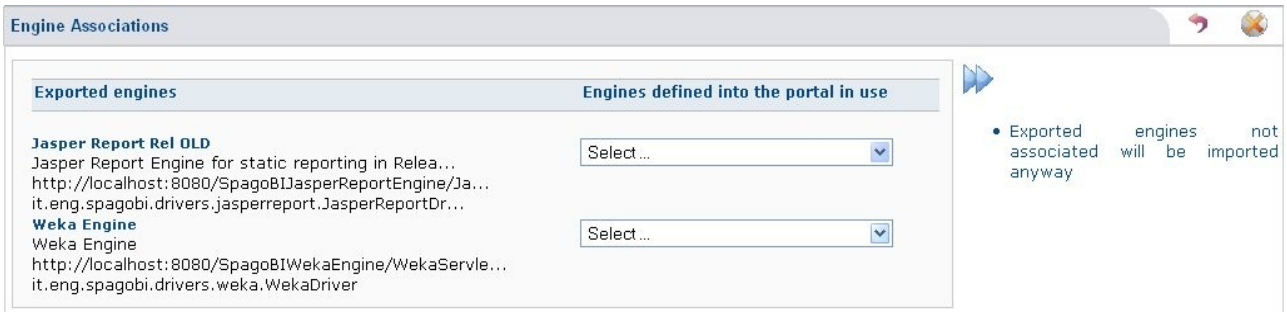

Figure 19 - Import engines association

The form shows the complete list of exported engines (engines associated to the exported documents) and, for each one of them, there's a combo which allows you to select the corresponding engine into the current system. If you don't provide an association the exported engine will be inserted anyway. Once complete the engine mapping you can select the next button (the right arrows), which will allow you to perform the connection mapping. This is the last mapping step and it's necessary because some of the exported list of values were associated to a database connection, which should be different form the ones of the current system

Figure 20 - Import connections association

Once again there is a description for each exported connection and a combo which allows you to associated a connection of the current system. This time, however, if you don't provide an association for each connection, the system shows you an error. In fact, it is not possible neither to insert an exported connection, nor to have some imported list of values connecting to an connection witch doesn't exist.. Once complete the connection mapping you can select the next button (the right arrows) which will complete the import operation.
During this latest phase the system checks if some exported objects conflicts with existing objects (for example if they have the same label or the same path). In case some conflicts are found, the sytem shows a complete list of all the conflicts. The user has the possibility to stop

the import, in order to resolve the conflicts and perform again the export operation, or continue, pressing the next button. If the user continues, the system doesn't import the metadata which causes the conflict, but replace it with the existing one.

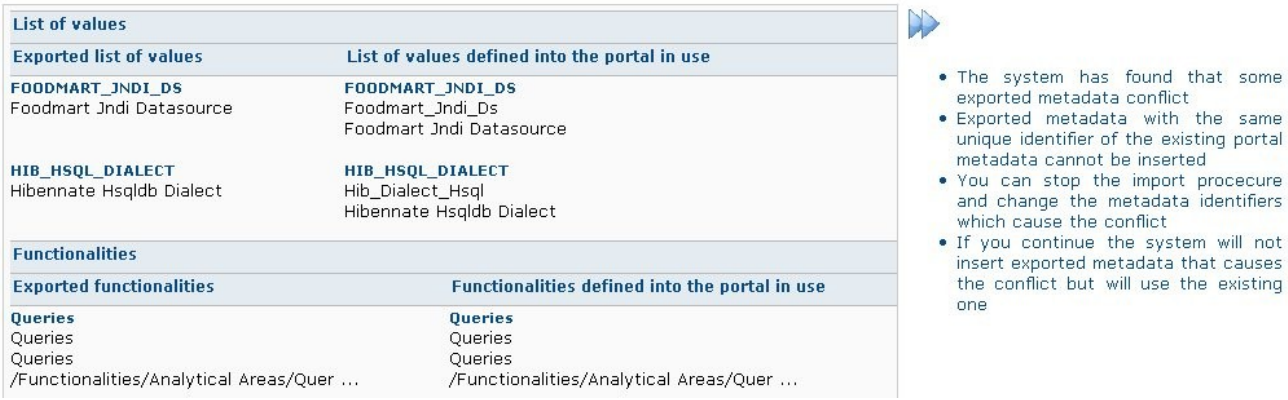

Figure 21 - Import conflicts

Once finished the import phase the user is redirect to the import / export main page. When displayed after an import operation the main page contains a 'Download log file' link, which allows to download a log file containing all the operation performed by the system.

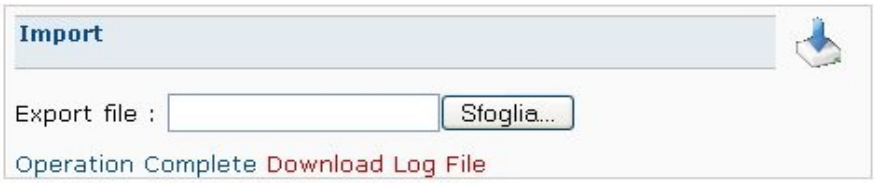

Figure 22 - Import operation complete

6.2. Administrator

The SpagoBI demo administrator (biadmin/biadmin user) has also the functionalities and documents administration portlet which permits him to perform the following operations:

- Managing the functional structure that classifies the analytical documents and configuring permissions on the functionalities;
- Maintaining the registered analytical documents.

Notice that the administrator manages the **structural configuration** of the platform

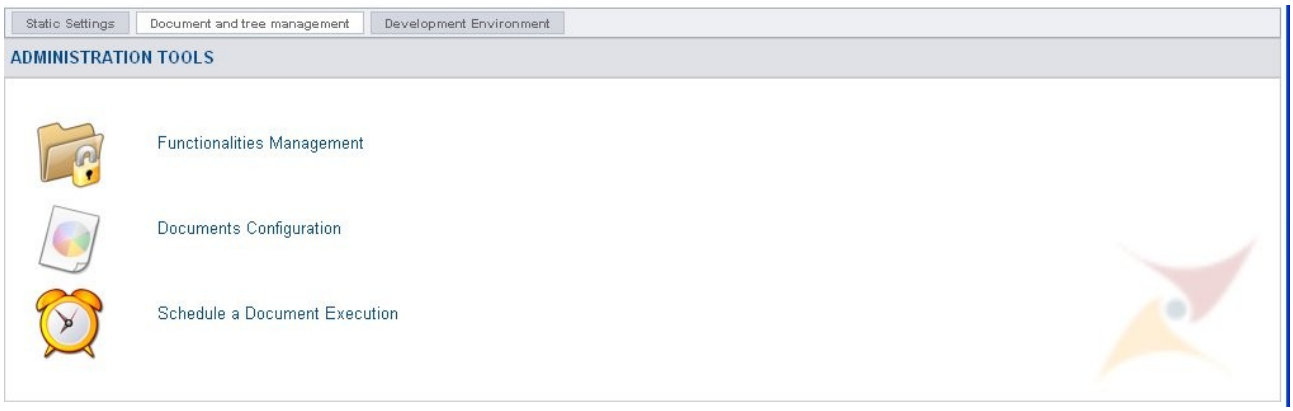

Figure 23 – Administration portlet

6.2.1. Functionalities Management

SpagoBI uses its own file system, named "**Functionalities Tree**", that allows to better organize documents by grouping them by folder: the access to those folders is regulated by roles permissions.
This multi-level hierarchical structure (Figure 24 – Functionalities Tree) can be created and modified exclusively by the administrator in the "Functionalities Management" area.

Notice that it is only possible to add a new folder to the root element, called *Functionalities*, but not to modify it.

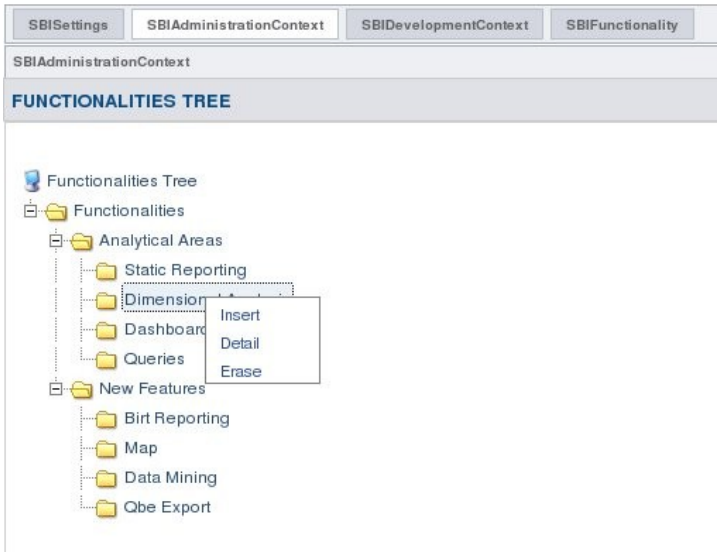

Figure 24 – Functionalities Tree

A list of possible actions can be visualized by clicking on a node of the *Functionalities Tree*.
The administrator can **Delete** an existing functionality, if this doesn't contain any sub-nodes. Moreover, he can create a new functionality. By choosing the **Insert** option, he can access a new page where he can fill in all required information. This new element will be child of the

selected one. Detailed information regarding an existing functionality can be displayed and modified by selecting the **Detail** option.

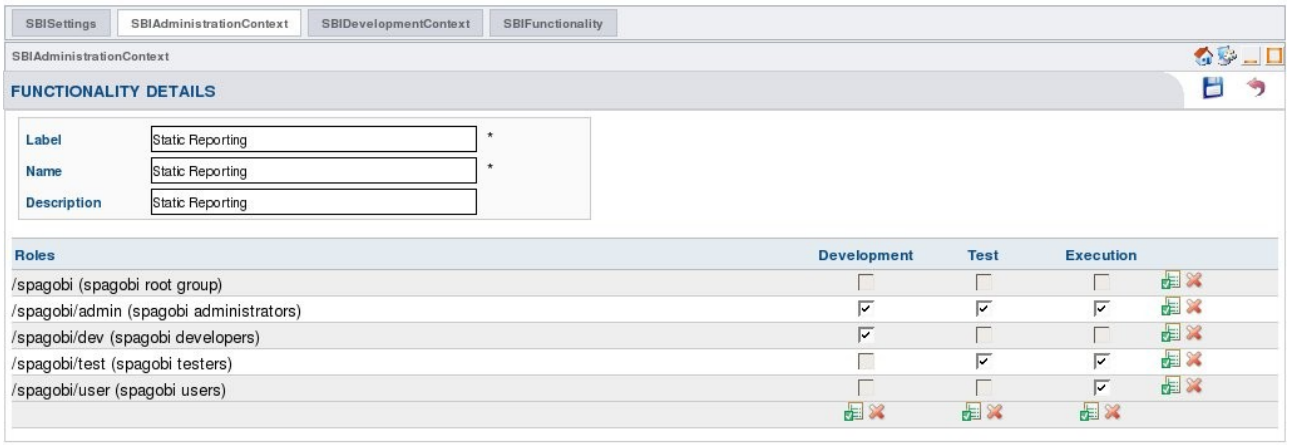

Figure 25 – Functionality details

Each folder is characterized by a name, a unique code and a optional brief description.
The list of the *Roles* allows the administrator to choose, for the selected functionality, whether to assign or remove the permissions required for the development, test or execution phase for each role.

For instance, checking the Development and the Test boxes of the "SpagoBI developers group", all users belonging to that group will only be able to develop and test documents contained into the selected functionality, but not to execute them when they are in Released state.

When we add or modify a functionality, all permissions that we can add or remove to each role follow a well defined policy. When we create a new functionality, by adding a new folder to the tree represented in the Figure 24 – Functionalities Tree, it inherits automatically all father's permissions. All permissions that the father doesn't have, *cannot be assigned neither to the children, nor to all the other descendants*.
According to this policy, if we add a child folder "Static Reporting Child 1" to "Static Reporting" folder, whose assigned permissions are represented in the Figure 25 – Functionality details, we have the following situation:

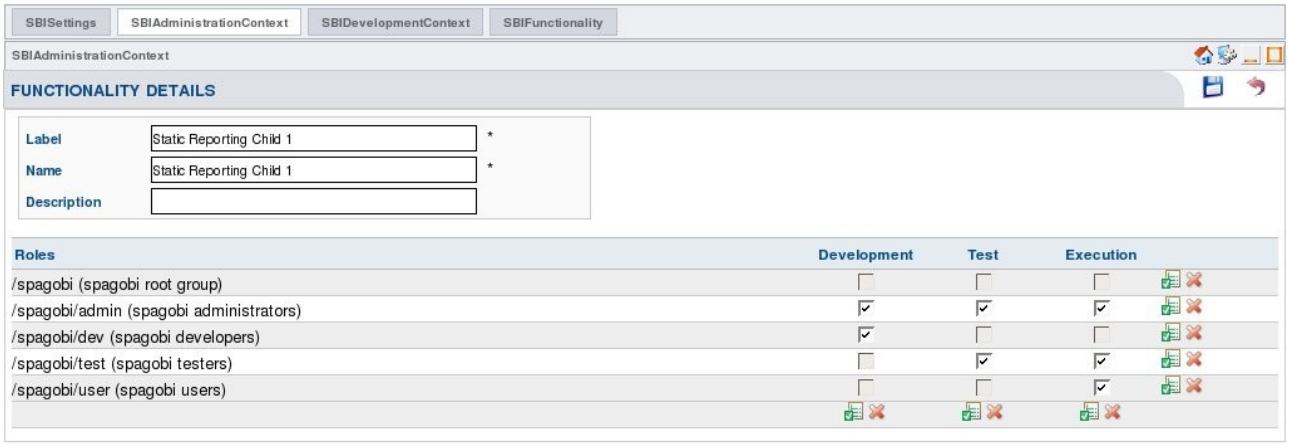

Figure 26 - Adding a new folder

As we can see, all Static Reporting functionality's roles are inherited, all the other ones are not assignable. Now, if we don't check any Execution permission for the Static Reporting Child 1, and after we add to it a new child functionality, for this one won't be able to select not only all permissions that we couldn't select for his father, but also the other two permission just revoked to the father itself.
Finally, the permissions management is configured in a way that we can never assign to a child functionality a permission that his father doesn't have, and this behaviour transfers itself from the root to the leaves nodes of the functionalities tree.

6.2.2. Document Configuration

A list of all the *Analytical Documents* registered in SpagoBI can be listed accessing to the *Documents Configuration* by the administrator. This area allows the administrator to manage the extraordinary maintenance of the documents.

Notice that the administrator is also a user and therefore he can execute all the released documents belonging to a folder on which he has the *Execution* permission.

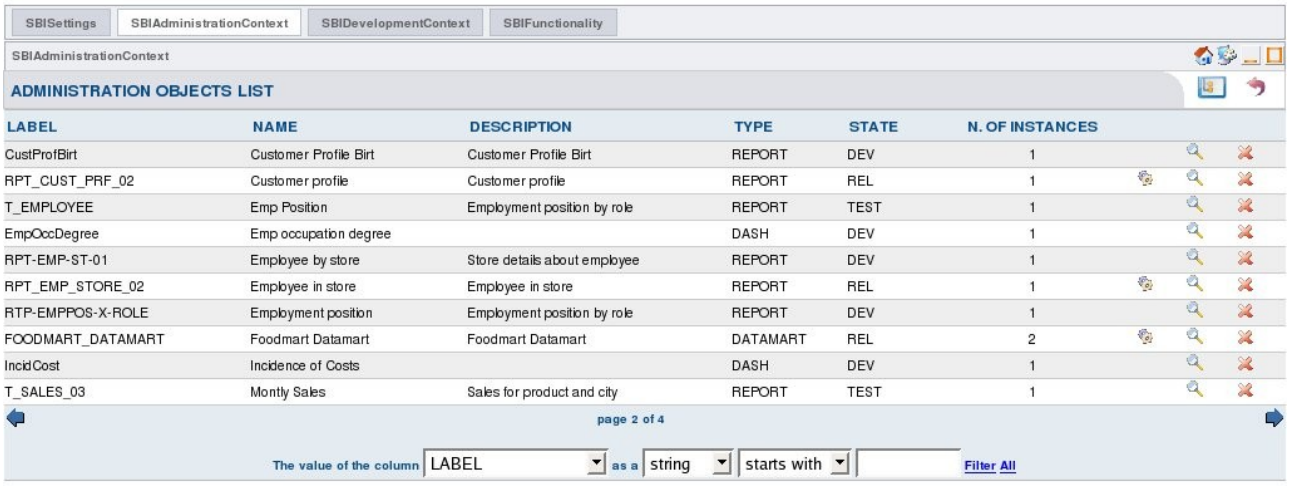

Figure 27 – Administration Objects List

Each document is described by a subset of its attributes, as follows:

- **Label**: the document unique identifier;
- **Name**: the document name;
- **Description**: a brief description of the document (optional);
- **Type**: this field shows if the document is a Report, an On-line analytical processing (OLAP), a Data Mining model, a Dashboard, etc.;
- **State**: this information indicates if the document must be developed (*Development*), tested (*Test*) or can be executed (*Released*). Moreover the document can also be *Suspended* if it cannot be executed anymore;
- **N. of instances**: since one document can be referenced in more than one folder, this is the folders number that contain the document.

The administrator has the *List View* but he can switch to the *Tree View* in order to have a list of documents grouped by functionality, clicking on the VIEW AS TREE icon. In the tree, each document has an icon, that represents the type, and a led, that represents the state: blue led stands for *Development*, yellow led stands for *Test*, green led stands for *Released*, red led stands for *Suspended*.

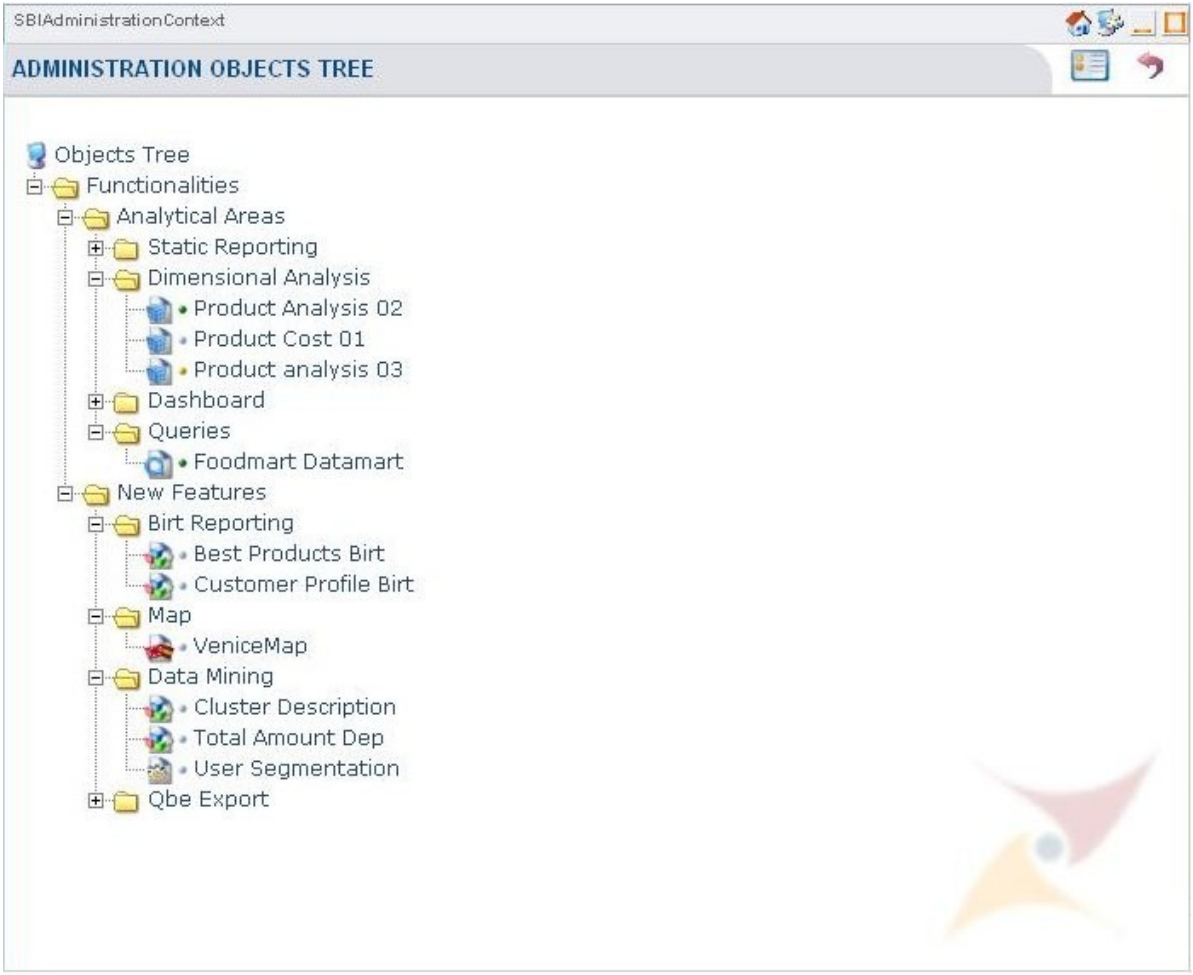

Figura 28 - Administration Objects Tree

When he deletes a document from the *List View* he deletes all the document instances he can see, while when he deletes a document from the *Tree View* he deletes only the selected instance.

The complete list of information can be seen in the *Document Details* page:

- **Engine**: the name of a registered engine that has to be used to execute the *Analytical Document*;
- **Criptable**: a flag indicating id the document is criptable;
- **Visible**: a flag indicationg if the document is visible when it is in Released state;
- **Template**: a file containing the model of the document to be created with an external application suitable for the specific type of the Analytical Document. On the right hand side of the page, in the "Template Version" table, all templates that have been selected for this document since it was created are listed. For each template, this list specifies the version identifier, the date when this selection occurred first and the name of the file. Through this view, the user will always be able to erase, download or select one of the listed templates.

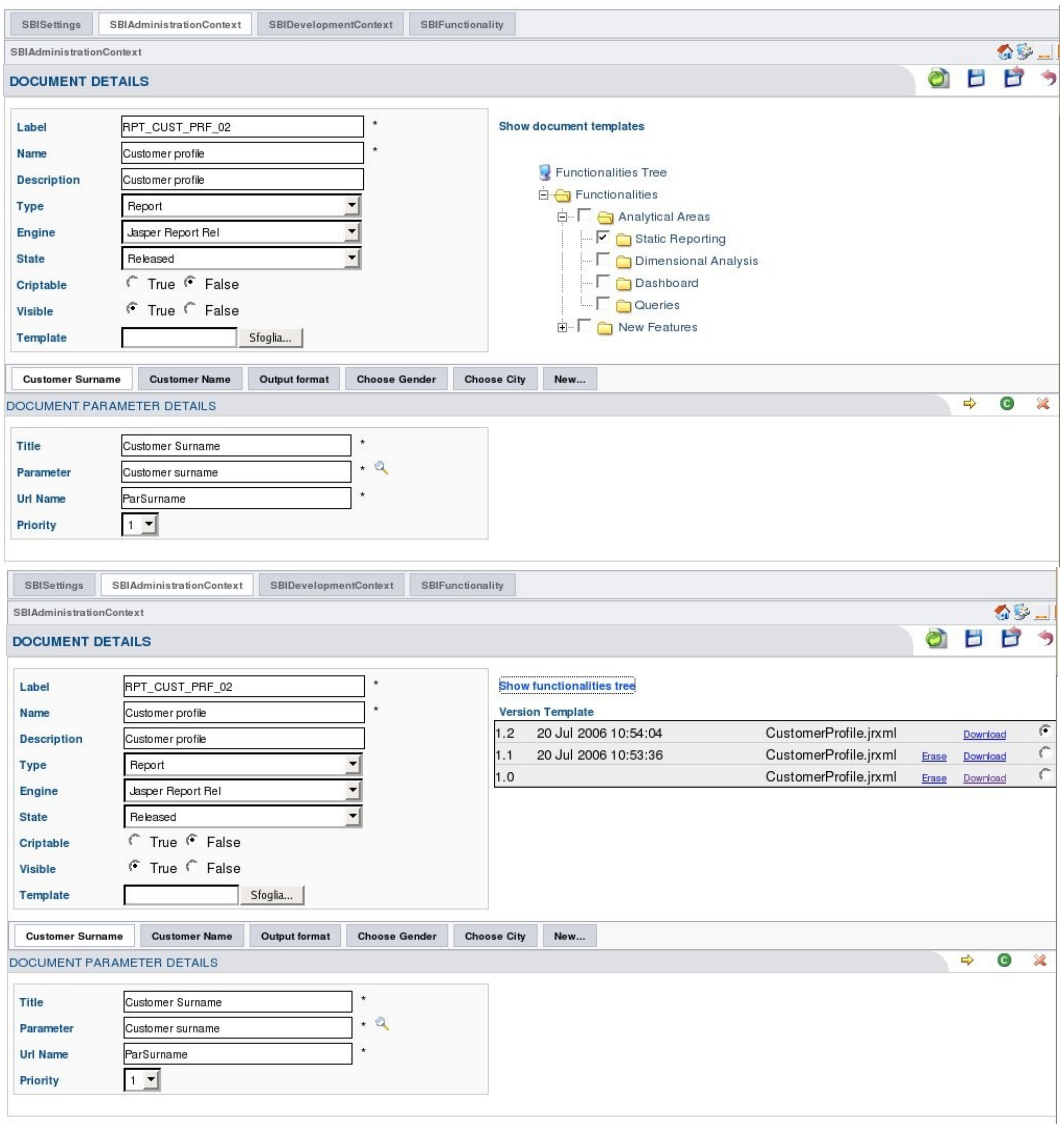

Figure 29 – Administration Object Details

In the lower part of the page you can see all the parameters applied to the document. In the *DOCUMENT PARAMETER DETAILS* section there are the following information:

- **Title**: the title of the document parameter;
- **Parameter**: the parameter that is applied to this document;
- **Url Name**: the key of the parameter: the URL for the document execution will contain an attribute with key specified by the content of this field and value specified by the parameter value;
- **Priority**: the document parameters are sorted by this number; clicking on the yellow arrows you can make a single step shift for the current document parameter.

You can switch from one parameter to another by clicking on the required element in the tab list.

The meaning of the field Parameter will be clarified in the next paragraph and then in the examples in chapter 5.

6.2.3.Schedule a Document Execution

SpagoBI allows the administrator users to schedule the execution of an Analitycal Document at a given time. The Document will be executed automatically and, based on the schedule configuration, the execution result can be stored as snapshot of the Analytical document.
A tree structure which contains all the Analytical Documents that can be scheduled (the ones in released state) is displayed when an administrator selects the 'Schedule a Document Execution' option.

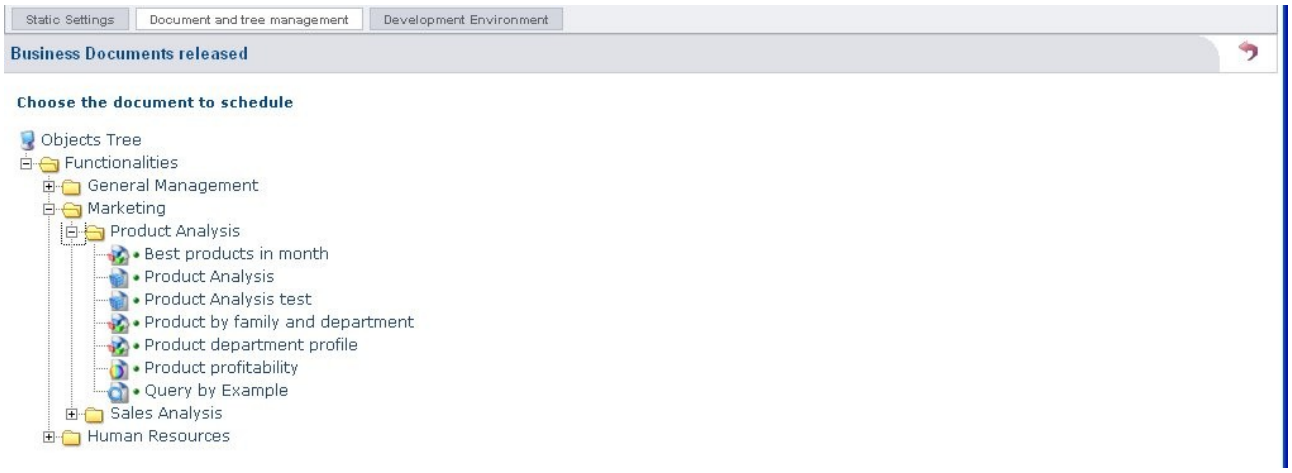

Figura 30 Tree of the documents that can be scheduled

Once the document to schedule has been selected from the tree, the administrator can view the list of the previously defined schedules. Then he can delete them, using the 'x' button of the list, change them using the hand len button of the list, or add a new one using the hand len button in the right corner of the title bar.

Figura 31 Schedule list of a document

If the administrator decides to change an existing schedule or to create a new one, he enters into the schedule detail page.

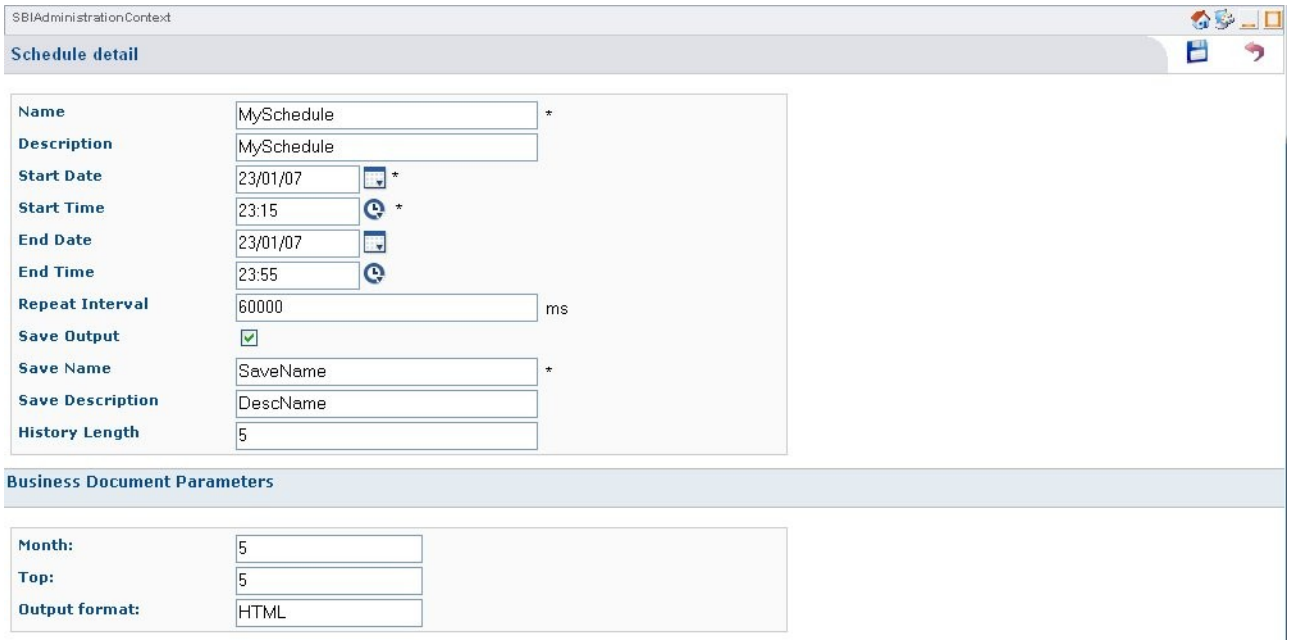

Figura 32 Detail of a schedule

A schedule is the definition of a time period. The automatic execution of the document can occur one or more time only between the initial and final instant of the time period defined. The execution can be repeated at regular time intervals.
The administrator can decide to store the result of the document execution. The result will be saved as a Document snapshot. The list of the document snapshots saved will apper to the users when they exec the document.
To control the number of snapshot saved for each document the administrator can define the maximun number of saving. When a new snapshot has to be saved, but the limit has been reached, the the first saved snapshot will be replaced by the new one. In this manner the list of snapshots can't increase without control.
The schedule detail page allows the administrator to define:

- Name: the name of the schedule
- Description: the description of the schedule
- Start Date: the initial date which identify the date lower boundary of the execution period
- Start Time: the initial time which identify the time lower boundary of the execution period
- End Date: the final date which identify the date upper boundary of the execution period
- End Time: the final time which identify the time upper boundary of the execution period
- Repeat Interval: the time interval between each execution.
- Save Output: save the output of the execution or not

Only if the administrator decides to store output he has to define also:

- Save Name: the name of the saved result (snapshot name)
- Save Description: the description of the saved result (snapshot description)
- History Length: the maximun number of snapshots saved

Below the schedule definition fields the detail page displays all the document parameter labels and the relative fields to assign a value to the corresponding parameter. Obviously this part is

variable because it is based on the document (different documents have different parameters). The mandatory parameters of the document must be filled, otherwise the automatic execution will not be able to produce the output

Based on the schedules defined, the platform executes automatically the documents and save the output (only if requested) as snapshot of the documents. When a User select a document for the execution, if the document has some snapshots saved, then the platform presents the snapshot list.

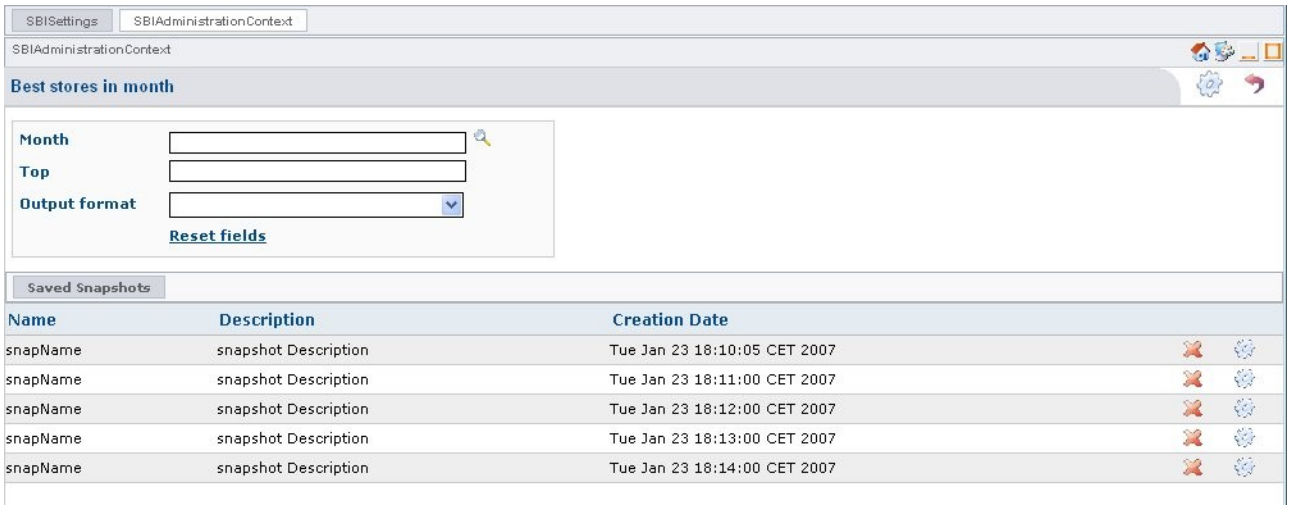

Figura 33 Example of snapshot list

The user can choose to launch a new execution of the document, filling the parameter values, or to view the output of a scheduled execution. (selecting the correspondingf hand-len button on the snapshot list)

6.3. Developer

☺ The developer's role is quite complex. This section explains its functionalities. In the chapter 5 you can find an example showing how the single functionalities work together for the right environment settings.

The developer (bidev/bidev user) main tasks are:

- to define the possible presentation and the preloading way (LOV – list of values) for the parameters;
- to define the validation rules (CHECK) for the input value;
- to create the parameters (PARAMETER) and to set up their behaviour rules associating LOV and CHECK to the user's roles;
- to register and to configure analytical documents, referring to the used parameters.

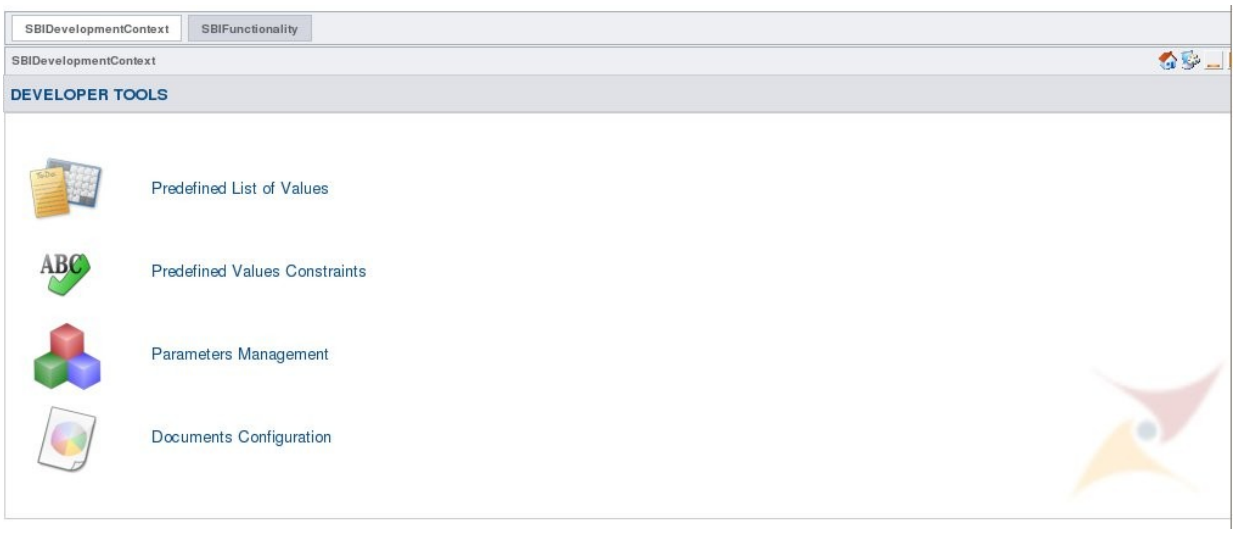

Figure 34 – Development portlet

Notice that the developer manages the **executive configuration** of the platform and only a few **structural configuration**.

Notice that at the start-up you have to create many LOV, CHECK and PARAMETER, but when the system runs regularly, you have to associate just the parameters already created to the new documents. You have to create new ones only for exceptions.

6.3.1. Predefined List of Value (LOV)

From the *Developer Tools* view it is possible to access the list of the *Predefined List of Value* (LOV). It is described by some identifying data (label, name and description) and by its Input Type.

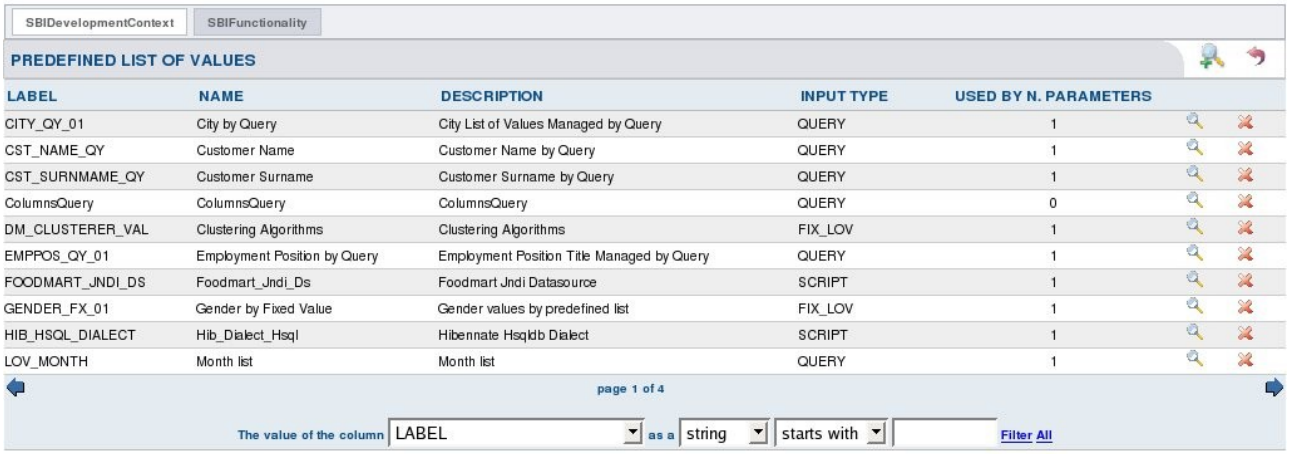

Figure 35 – Predefined List of Values

Accessing the details page, general information are displayed:

- **Label**: LOV unique identifier;
- **Name**: LOV name;
- **Description**: brief description of the LOV (optional);
- **Input Type:** three different types of LOV are admitted allowing:
  - **Query:** the database retrieval of all the selected values;
  - **Fixed LOV**: the organization of an arbitrary value list;
  - **Script:** the registration of a Groovy script delegated to the recovering of all the values to suggest.

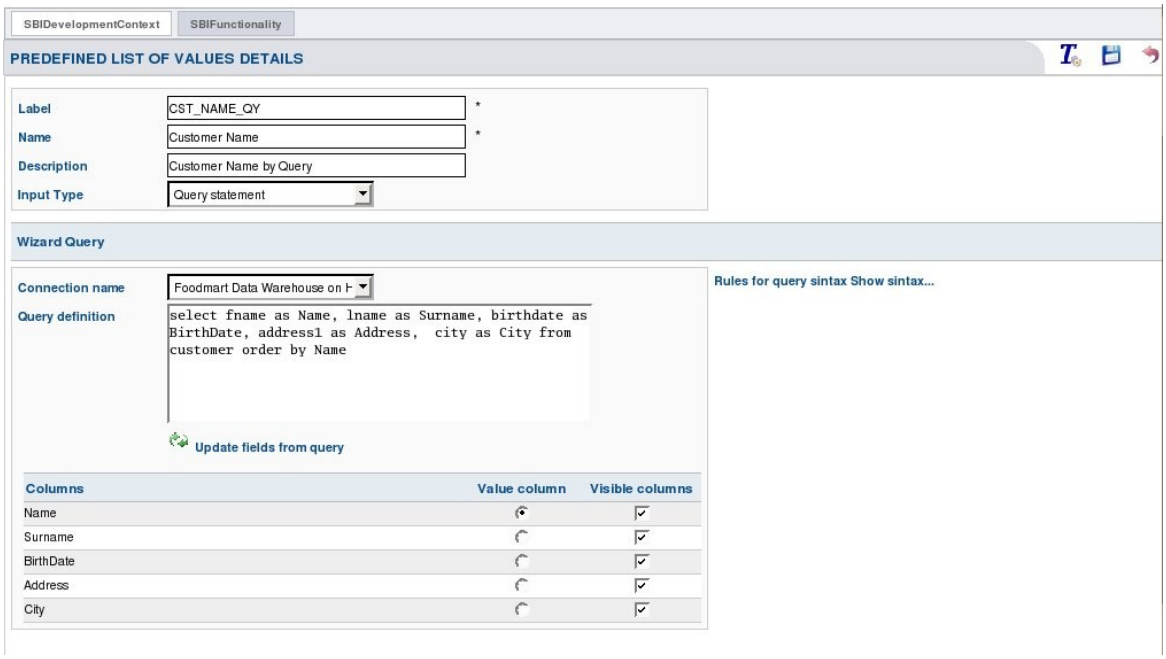

Figure 36 - Predefined List of Values Details

According to the selected typology, in the bottom of the page a wizard is available in order to ease its specific composition.

In the *Query Statement* case (the picture example above) the required information are:

- **Connection Name:** logic identifier of the database source;
- **Query Definition:** the SQL statement;
- **Value Column:** name of the unique column of the dataset containing values that will be return as a result of the parameter;
- **Visible Columns:** name of the columns of the dataset that will be displayed;

In the *Fixed Values case,* you can create a table of pairs (Name,Value) simply by filling the *New item name* and the *New item value* fields and clicking on the ADD icon. In order to erase an existing pair you can use the icon on the corresponding row of the list at the bottom of the window.

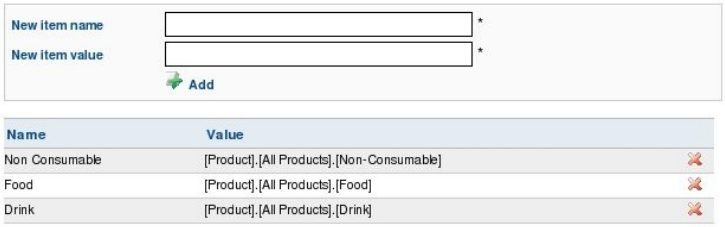

Figure 37 – Fixed values wizard

In the *Script* case you have to write the Groovy script to be executed at run time.

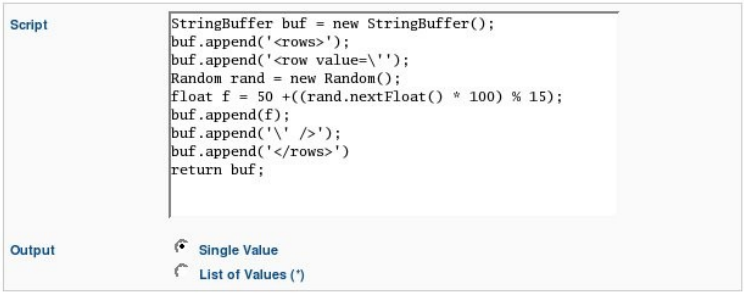

Figure 38 – Script wizard

6.3.2. PREDEFINED VALUES CONSTRAINTS

The developer can register some typologies of formal controls applying to the values inserted in the documents activations phase.

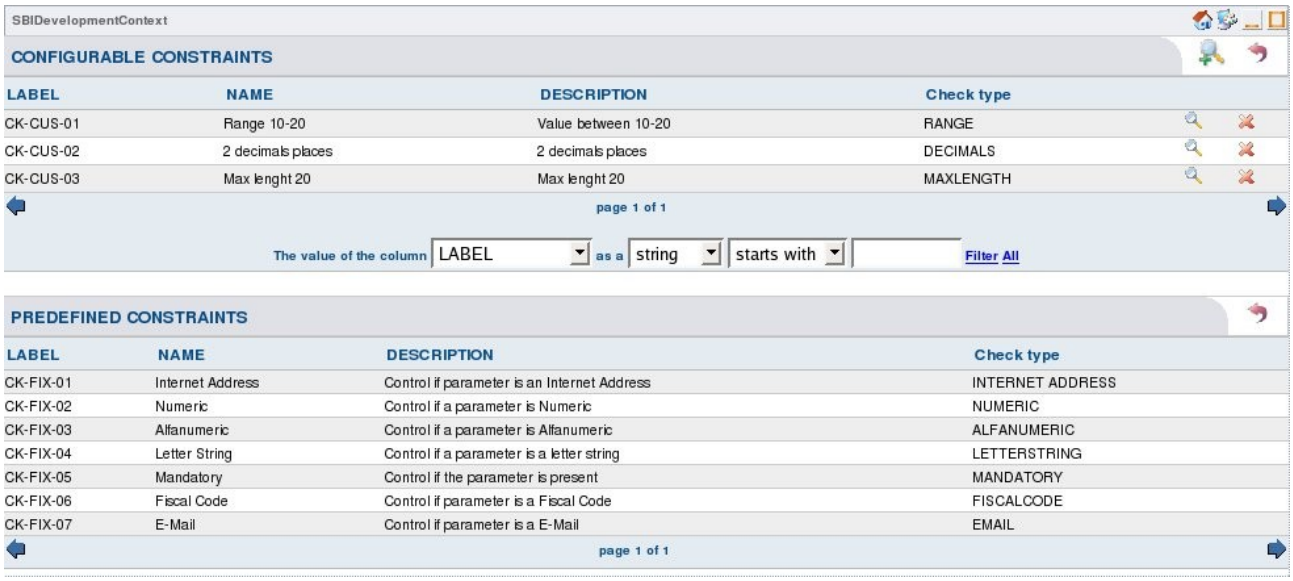

Figure 39 - Configurable Constraints and Predefined Constraints

At the bottom of the displayed page there is a list of *Predefined Constraints* that cannot be modified. On the top of the page it is possible to create a set of *Configurable constraints*, simply clicking on the INSERT icon and entering the *Constraint Details* page.
The developer can add new configurable controls, erase the existing ones or access their detailed information in order to modify them.

Figure 40 - Constraint Details

Every constraint is identified by a unique label, a name and a brief and optional description.
In addition to some identification data (label, name and description) it is possible to configure some different control typologies:

- **date**: date format control;
- **regexp**: control through regular expressions;
- **min length**: minimum number of characters for the inserted values;
- **range**: to control a value included into two limits;
- **decimal**: decimal digits control;
- **max length**: maximum number of characters for the inserted values.

The developer can select a *Check Type* from the list and filling in the required values. Each constraint can have just one *Check Type*.

6.3.3. Parameters Management

SpagoBI handles the parameters in term of autonomous entities, each one with its own behaviour based on users' roles. So it is possible to associate to them different presentation (LOV) and validation (CHECK) rules.
The list view allows the developer to add new parameters, to erase the existing ones or to access their detailed information in order to modify them.

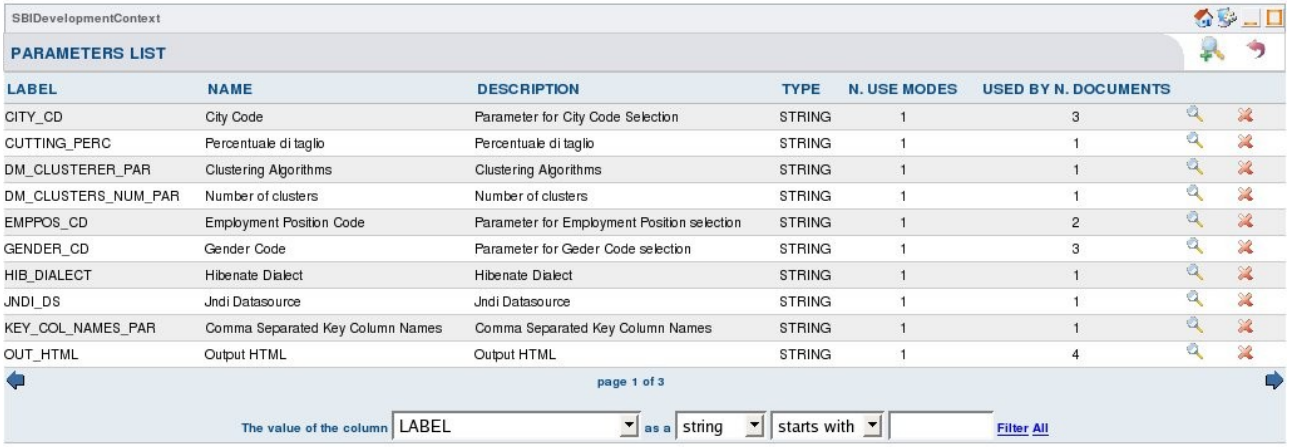

Figure 41 – Parameters list

A parameter is identified by the following information:

- **Label**: unique identifier of the parameter;
- **Name**: name of the parameter;
- **Description**: brief description of the parameter;
- **Type**: define if this parameter is a date, a number or a string;
- **Functional**: define if the parameter is functional or not: a functional parameter is a business logic parameter (examples: city, customer, year of analysis ...) while a non functional parameter is independent from the business logic (examples: output format, database connection ...).

To insert a new parameter the developer has to fill the following form:

Figure 42 – New parameter form

When entering instead an existing parameter detail page he can see also the use modes of the parameter (the tab list in the following picture):

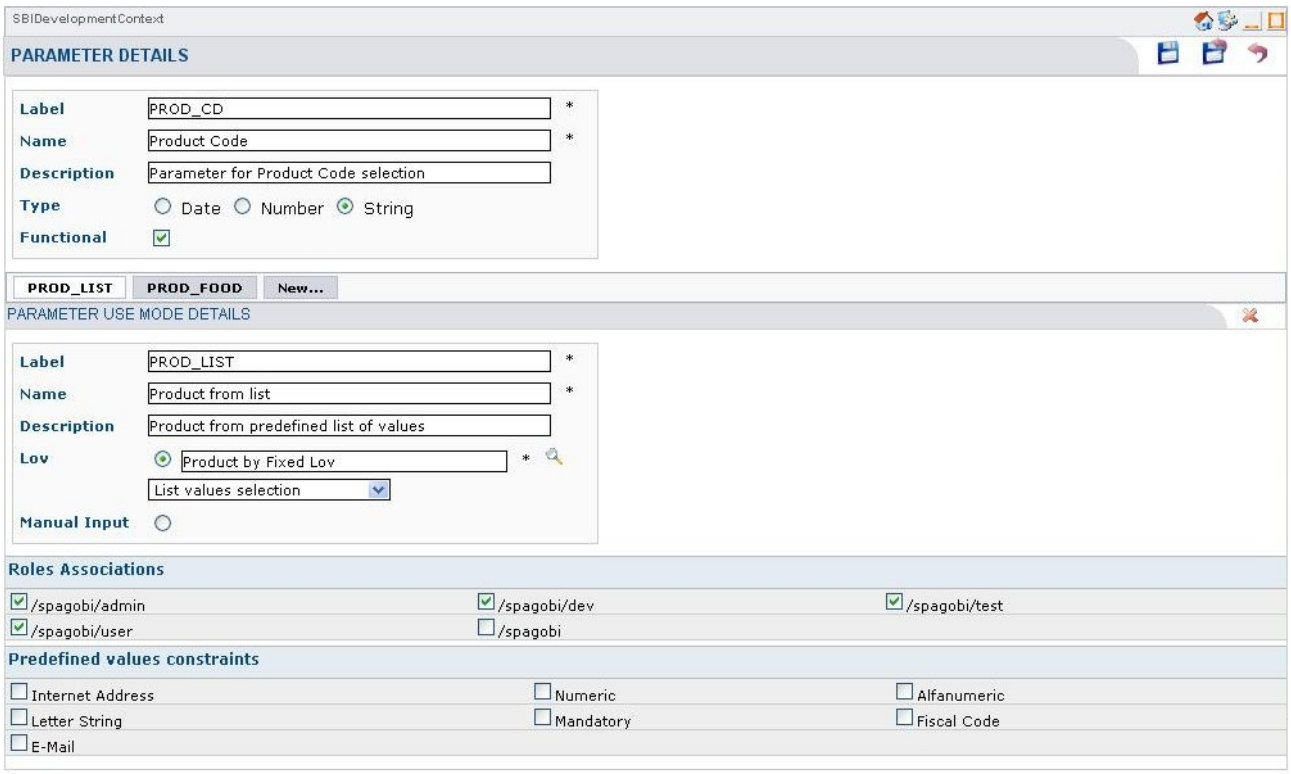

Figure 43 – Parameter details

As described before, each parameter can manage different user roles simply by assigning a specific way to collect data and specific constraints to validate the final input. This means that each role must be assigned to a specific *Use Mode*.

Each Use Mode requires the following information:

- **Label**: *Use Mode* unique identifiers;
- **Name**: *Use Mode* name;
- **Description**: brief description of the *Use Mode*;
- **Lov**: it is possible to select just one way to collect data from the list of available LOVs. Click on the LOV LOOKUP image and then select a Predefined List of Value from the lookup page (to create a new *List of Values* please refer to the Predefined List of Value paragraph); this is in alternative with the manual input; then you have to choose the modality of visualization of the parameter between: lookup list values selection (using this selection type the user will be able to choose only one value for the parameter), lookup checklist values selection (the user will be able to choose more than one parameter values(1)) or combobox (one parameter value);
- **Manual input**: as an alternative to the lov association, the modality can be manual input type, i.e. the parameter value will be filled manually by the user;
- **Roles Association**: list of the roles associated to this *Use Mode*. A role cannot be associated to more than one *Use Mode;* anyway a role may not be associated to any Parameter *Use Mode*. In the second case a user having just that role will not be able to execute a document using this Parameter;
- **Predefined Values Contraints:** it is possible to assign zero, one or more constraints selecting them from the list of available constraints.

[1] Multi values parameters are useful only using SpagoBIJasperReportEngine (see Report); all other engines will consider only the first value.

6.3.4. Document configuration

The user, simply by clicking on *Documents Configuration* from the *Developer Tools* page, can display the *Development Object List.* This page catalogues all the *Analytical Documents* with a *Development* state or with a *Released* state. They have to be contained in a folder for which the user has a role authorized respectively for *Development* and for *Execution*.

Please notice that every new document will have the Development state. For a better comprehension, please refer to the Analytical Document life-cycle section.

Each document is described by a unique label, a name, a description and a type (report, OLAP, etc.).

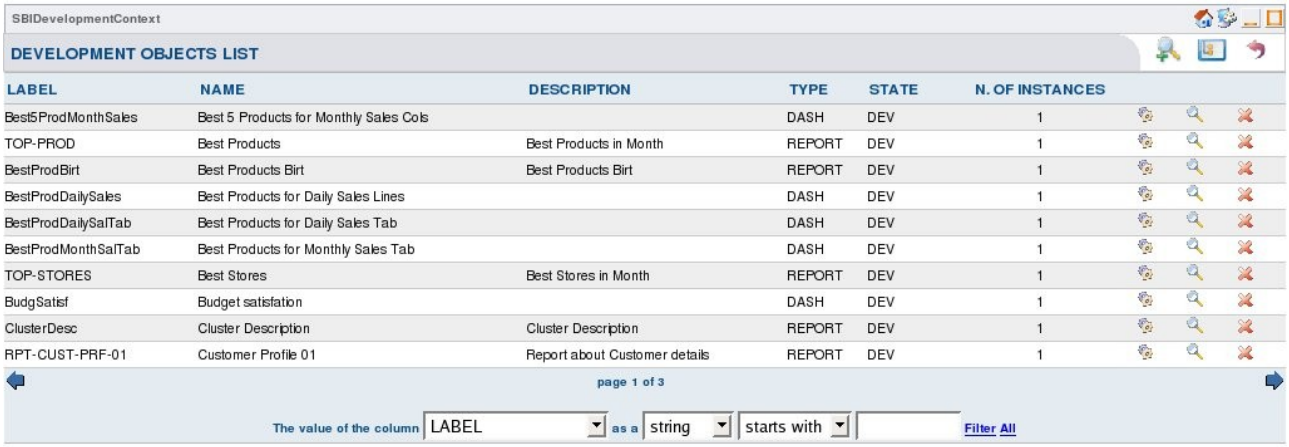

Figure 44 – Analytical documents list

The *Documents Details* page displays a complete list of attributes:

- **Label**: document unique identifier;
- **Name**: document name;
- **Description**: brief description of the document (optional);
- **Type**: document type (report, OLAP, Data mining model, Dashboard, etc.);
- **Engine**: engine that will be used to execute the document. The available engines are registered by the administrator (see Engines Configuration paragraph);
- **Criptable**: flag indicating if the document is criptable;
- **Visible**: flag indicating if the document is visible when it is in Released state;
- **Template**: file containing the model of the document. It has to be created with an external application suitable for the specific type of Analytical Document.

When the developer inserts a new document he has to indicate the **Parent folder** under which the document will be created: this can be set by selecting the corresponding check box on the *Functionality Tree* displayed on the right hand side of the window.

The developer can see only the details of the documents that are in *Development* state, so the **State** field (visible by the administrator) is hidden.

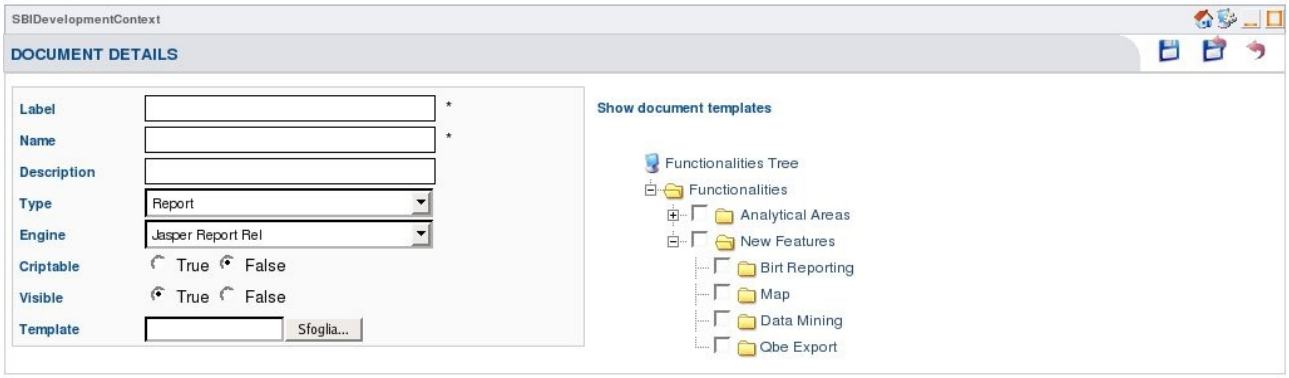

Figure 45 – Analytical document details (new document)

Multiple selection of the parent folder is allowed.

When the developer sees an existing document and selects the DETAILS icon, additional elements will be displayed:

- **Detailed information** (Top-left side): already explained data; they are the same view of the document creation phase;
- **Version template list** (Top-right side, chicking on "Show document templates"): every time the developer uploads a new template it will be set as the current default and will be added to this list specifying the version identifier, the date when this selection occurred first and the file name. By means of this view the user will always be able to erase, download or select one of the listed templates;
- **Functionalities tree** (Top-right side, chicking on "Show functionalities tree"): the functionalities tree, where functionalities containing the current document are checked. All the functionalities on which the user has the development permission can be checked and unchecked;
- **Parameters tabs** (bottom side): there is one tab for each parameter associated to the Analytical document. An additional tab (*New …*) creating a new association is provided.

In every parameter tab (for *Document Parameter Details*) the user is required to fill in the following information:

- **Title**: Document Parameter name;
- **Parameter**: clicking on the PARAMETERS LIST icon a lookup page for the available parameters list will be displayed;
- **URL Name**: parameter name on the document execution URL. This must match the corresponding parameter belonging to the template;
- **Priority**: the document parameters are sorted by this number; clicking on the yellow arrows you can make a single step shift for the current document parameter.

The developer can switch from one parameter to another by clicking on the required element in the tab list and can insert new parameters by clicking on the "*New…*" tab and fill in the form.

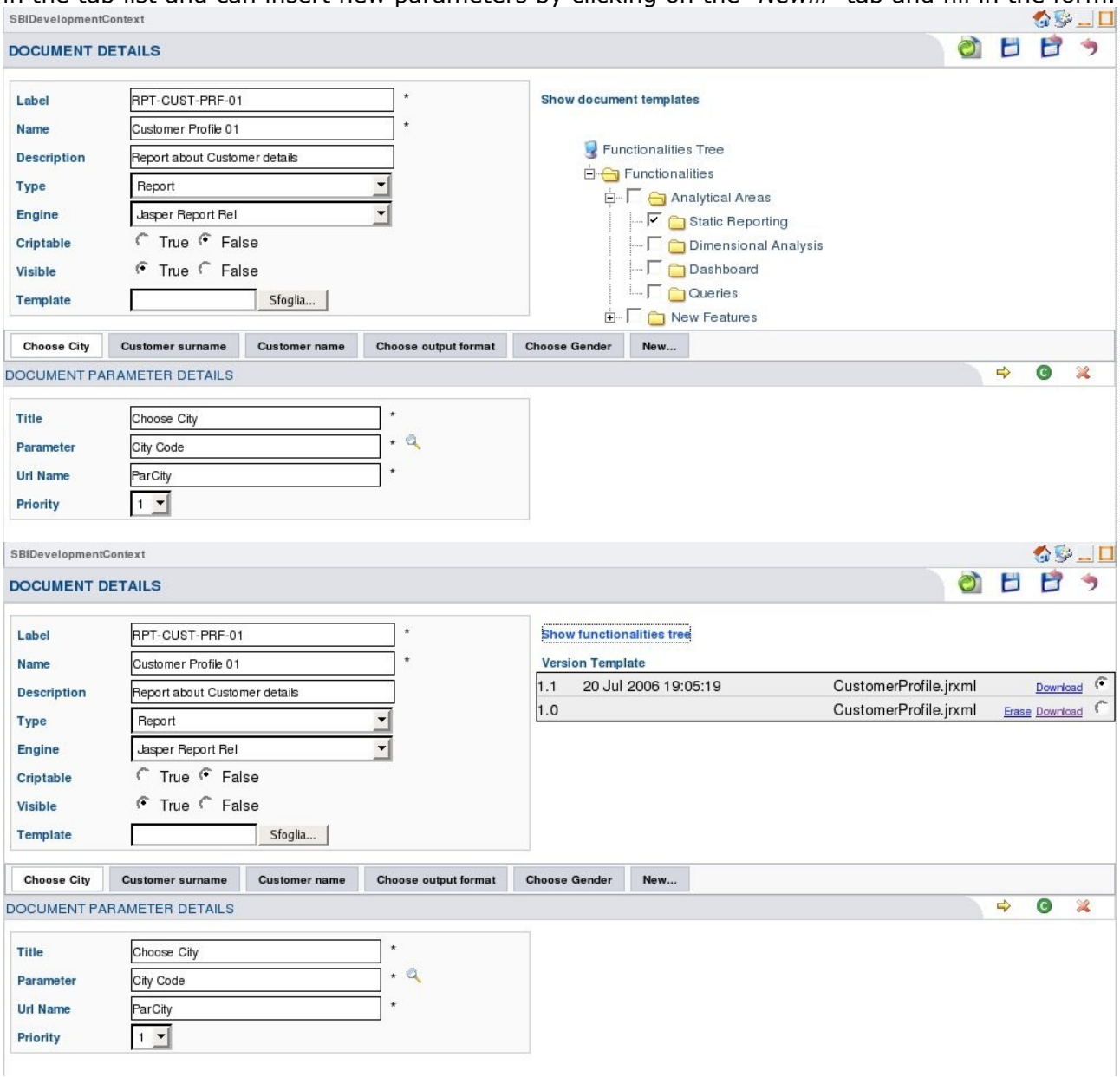

Figure 46 – Analytical document details (existing document)

Notice that during this phase it is not necessary to specify anymore about users' roles because they are completely managed through the visibility rules of the functional tree-view and through the behavioural parameters' description.

6.3.5. Parameters correlation

Generally speaking two different parameters are saperated and indipendent entities. In the context of a document two different parameters can be correlated instead: this means that the collection of the possible values of the former (the dependent parameter) is restricted by the value of the latter (the "father" parameter).
Since in SpagoBI the parameters can have different modalities, the actual correlation is between one parameter modality and the father parameter (no matter its own modalities, since only its final value is considered).
Notice that the father parameter is only one: it is not possible to establish a correlation with more that one father parameter; it is possible instead to establish cascading correlations (one parameter is dependent by a father parameter which depends by a third parameter and so on).
The correlation is not possible obviously if the dependent modality is single value type; it must be a multi-value modality. At the time being only modalities associated to queries and with lookup list (or check list) visualization mode can be put in correlation, but in the future may be it will be possible to correlate all modalities that produce a list of values.

In order to configure a correlation, within a document detail page, click on the dependent parameter detail and than click on the "Correlation" button; in the following page you will see a list containing all the modalities that can be put into correlation (i.e. all the modalities of the parameter that are associated to a query and with lookup list, or check list, visualization mode).
In the first column of the table you can check the modalities that you want to be correlated, then: since the query associated to a modality has one or more columns, you can set the column to be filtered by the father parameter and the filter type.
At last you have to choice the father parameter: obviously the possible parameters are all the other document parameters.
Notice that if the father parameter is String type the filter will make a literal filtering, if the father parameter is Number type the filter will make a numerical filtering and if the father parameter is Date type the filter will make a calendar filtering: this means that the column to be filtered is assumed to be of the same type (String, Number or Date) of the value of the father parameter.

If the father parameter is multi-values, the dependent parameter value satisfies the dependency if the filtering condition is satisfied with at least one father parameter value. Since the filter operators "<", ">", "≤", "≥" have no sense in case of multi-values father parameter, in such cases the dependency is ignored.

Example:
The parameter "Customer surname" is correlated with the "Choose City" as in figure.

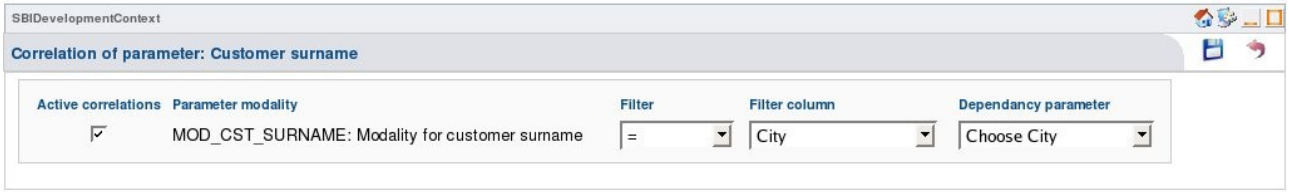

Figure 47 - Correlation configuration example

When the document is executed the possible values of the dependent parameter ("Customer surname") will be the customers living in the city (or the cities, in case of multi-values father parameter) specified by the father parameter ("Choose City").
The page for the document parameters selection remembers you all the existing correlations, as shown in the figure below (the father parameter is showed when the mouse is over the blue arrows).

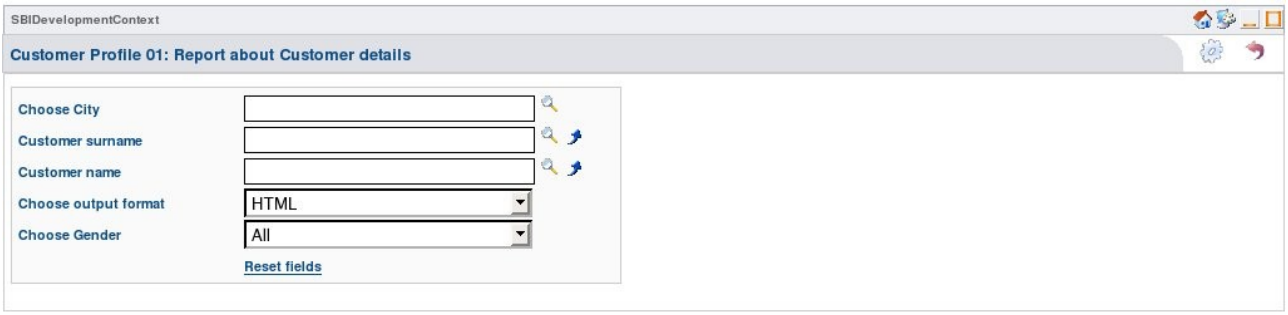

Figure 48 - Parameters selection page with correlated parameters

In case the father parameter was not valued, the dependent parameter ignores the correlation and produces the complete collection of possible values.

6.3.6. Links

In SpagoBI some documents can be correlated (linked): the kind of corrlation depens on the type of document.

At the time being SpagoBI supports two types of links:

1. subreports (master – detail report documents using JasperReport engine);
2. Data mining model – presentation report document.

Subreports

Subreports are reports that can be imported as building blocks into other more complex reports. Subreports allow to modularize the process of report definition. Modularization of reports can be very useful to simplify report template file in particular in the case of reports that have a master-detail structure.

From now on we will refer to a report that uses subreports as a *master report*. The goal of this section is to explain how to insert a master report into SpagoBI environement and make it works. The definition of a master report's template is an advanced JasperReports functionality and goes beyond the scope of this short introduction. More infos on this topic can be found into the JasperReport User Guide.

In order to insert a master report into SpagoBI environment you have to insert its subreports in the system first. A subreport it's just a normal report included in some other reports. There

is no difference between the insertion procedure of a subreport and the insertion procedure of a normal report just because there is no difference between subreports and normal reports (i.e every report can be used as subreport).

Once all the subreports used by the master report you want to use have been inserted into the system you can also insert the master report. In order to execute the master report at the end of the insertion procedure without any errors it is important to verify that its template file points to subreports included just by name (i.e. *subreport1.jasper*) and not by relative or absolute path (i.e. */myreports/subreport1.jasper*).

```
<subreport  isUsingCache="true">
    <reportElement
        mode="Opaque"
        x="0"
        y="48"
        width="782"
        height="20"
        forecolor="#000000"
        backcolor="#FFFFFF"
        key="subreport-1"
        stretchType="NoStretch"
        positionType="FixRelativeToTop"
        isPrintRepeatedValues="true"
        isRemoveLineWhenBlank="false"
        isPrintInFirstWholeBand="false"
        isPrintWhenDetailOverflows="false"/>
    <subreportParameter  name="Year">
        <subreportParameterExpression><![CDATA[$P{Year}]]></subreportParameterExpression>
    </subreportParameter>
    <subreportParameter  name="MonthName">
        <subreportParameterExpression><![CDATA[$F{MONTHNAME}]]></subreportParameterExpression>
    </subreportParameter>
    <connectionExpression><![CDATA[$P{REPORT_CONNECTION}]]></connectionExpression>
    <subreportExpression  class="java.lang.String"><![CDATA["subreport.jasper"]]></subreportExpression>
</subreport>
```

Figure 49 - Subreport definition in master report template

Once you have checked this you can proceed with the insertion procedure of the master report. Also this procedure is very similar to the standard insertion procedure of a normal report. The only difference is that you have to specify which reports, among already inserted ones, your master report will include. In order to specify this list of subreports you have to click on the subreport button in the document detail page (Figure 50 - Links configuration button) and than check from the list shown the report included by your master report (Figure 51 - Links list).

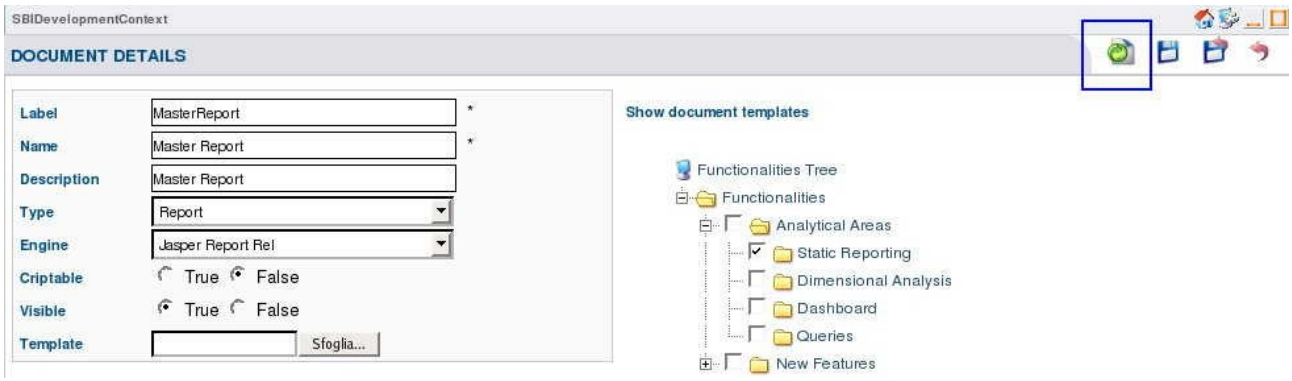

Figure 50 - Links configuration button

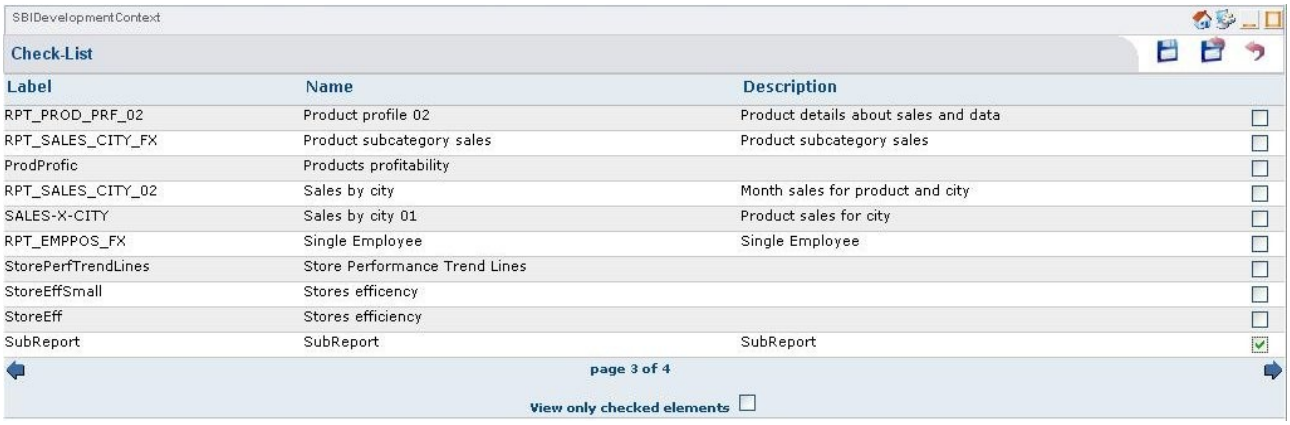

Figure 51 - Links list

In this page there is also the possibility to see only the checked documents by clicking on the checkbox below.

Data mining model – presentation report document

A data mining model in SpagoBI is actually an asynchronous process that produces some results. The presentation of the results is delegated to another report document (one or more) that must be executed only when data mining process is completed.

The result presentation report must be configured as a link on the data mining document configuration page. The user has to click on the "Links" button and then check the report document.

6.3.7. Documents visibility

Checking visible properties into the document detail page (Figure 52 - Document visibility field) to false make the document not visible in the object tree view when the document itself

passes from the developing state to the testing or releasing state. This option is useful when a document is used always as a building block in the definition of more complex documents and it has no sense to show it by its own (i.e. Subreport).

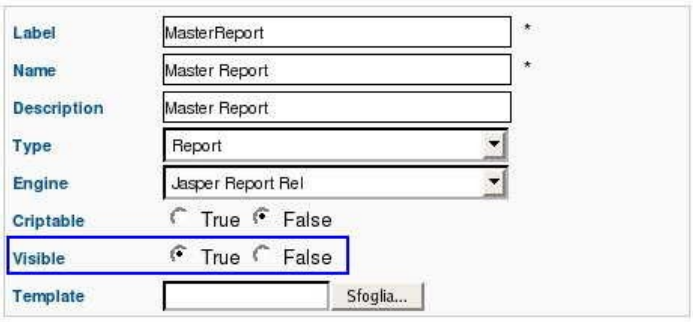

Figure 52 - Document visibility field

6.4. Tester

The tester (bitest/bitest user) main tasks are:

- validating the produced Analytical Document to simulate all its predefined roles;
- updating the Document state to release the documents that becomes available for the end-user.

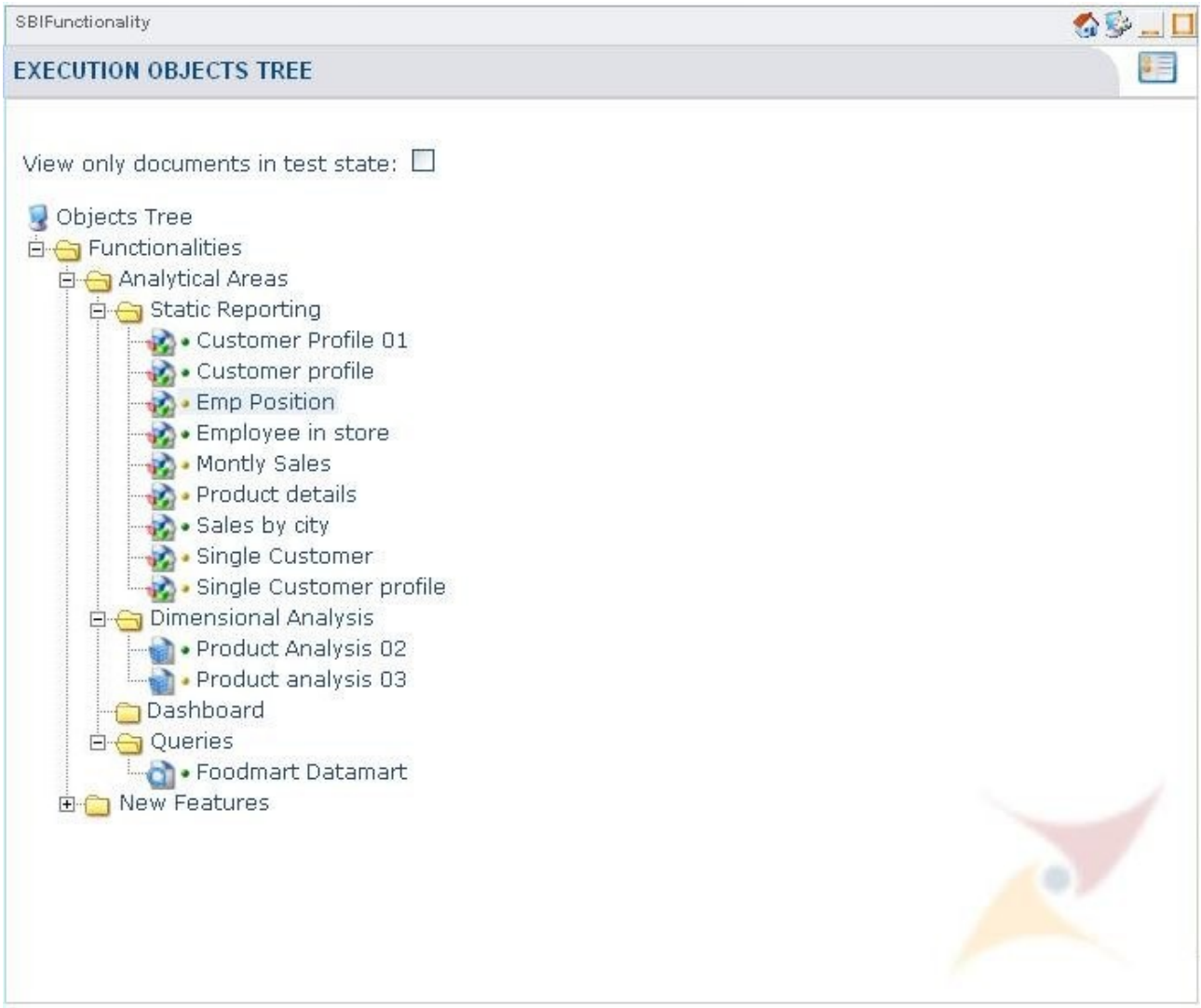

Figure 53 – Tester's documents tree

This type of user must verify the formal correctness of the registered documents and check if the documents in a test state works correctly and if they fulfil the requirements.
By means of the functional tree-view, this page lists all documents having *Test* or *Released* as current state (a document in *Test* state has a yellow led, a document in *Released* state has a green led). They belong to a folder for which the user has at least one role with a required permission respectively for Test and Execution.

For a better understanding of the Security Policy please refer to the Document organization and to the Security Policy paragraph.

The Tester can only execute a specific document by selecting it from the *Functionalities Tree* or from the correspondent list view.

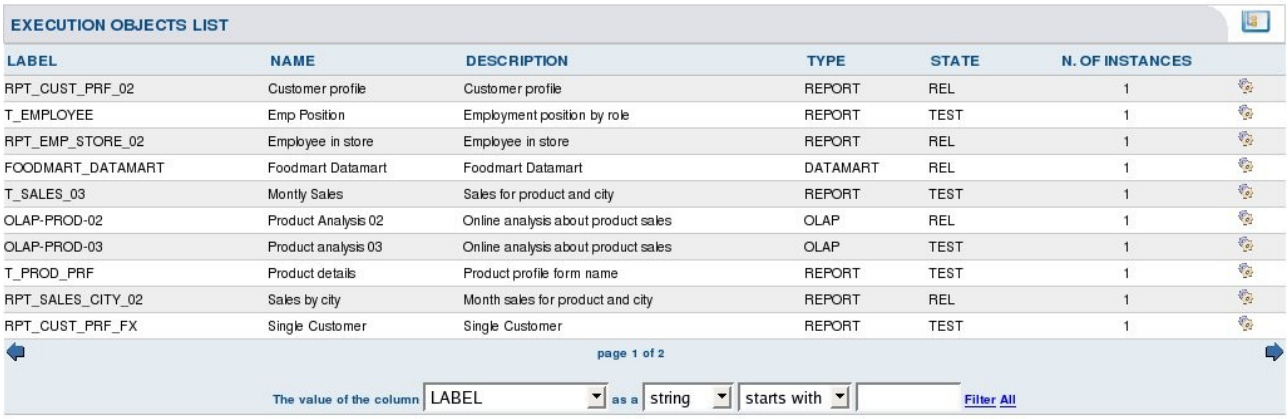

Figure 54 – Tester's documents list

If the selected document is in a *Test* state with a parameter associated to different *Use Modes*, the user has to choose a role from the *Role* combo box in the *Select Role For Execution* page.

Figure 55 – Role selection page

Notice that the list includes all the system roles available for the document and not only the ones belonging to the *Tester*. This means that the *Tester* will be able to test the different behaviour of the *Analytical Document* in relation to every user's role. No matter whether he owns the role or not. Instead, if the document is in *Released* state, the list includes only his own roles (in case he is associated with more than one role).

By clicking on the EXECUTION icon, the *Analytical Document* will be run and, if necessary, a page for the input parameter will be displayed allowing the user to insert the required information (from and according to the Analytical Document configuration).

Figure 56 – Parameters page

The page is under a revision phase.

When a parameter is defined as a *Query* type, the corresponding input field becomes a *Lookup* table where you can choose your value.

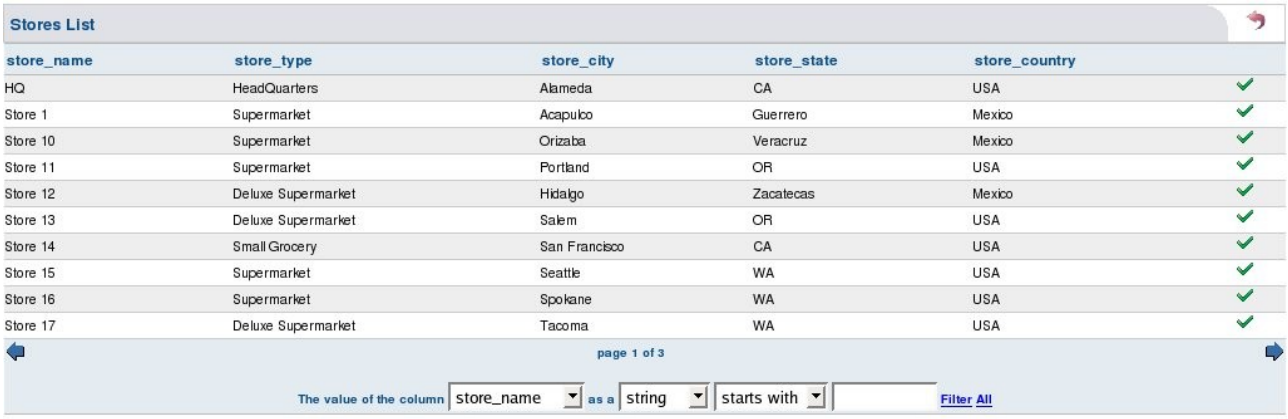

Figure 57 – Lookup for parameter input value

The list pages can be turn over using the two arrows on the bottom row. The desired value can be selected by clicking on the corresponding icon. Moreover, to ease the user to find the required data, it is possible to filter the list.
When the parameter selection is completed the user can execute the document by clicking on the icon.

Figure 58 – Final execution

Now the test user can update the document state to *Release*, if all the required tests worked correctly; otherwise to *Development*.

Moreover, he can click on the BACK icon in order to execute a different test using a different role.

6.5. END-USER

The end-user (biuser/biuser user) works with the **Analytical Portal** made by the Business Intelligence designer and developer.
The on-line demo is an example of an analytical portal whose purpose is to let you see the basic elements for the building of your analytical portal.
For example, from the menu of the Demo portal, you can access many sections:

- **Home**: many portlets which combine punctual views of the performance indicators with synthetic reports.
- **Navigation**: a free navigation on the functional tree of the documents.
- **Instanced Reports**: a page divided into subsections for the presentation of single reports already instanced (not parametrics).
- **Parametric Reports**: a page divided into subsections for the parametric activation of the reports with default values.
- **Olap**: a multidimensional analysis model.
- **Dashboard**: a synthetic, static and dynamic presentation of historical and current series with especially interests.
- **Dynamic Dashboard**: a synthetic and dynamic presentation of the performance indicators values to be monitored at fixed time intervals.
- **QbE** (Query by Example): the module for a free and visual inquiry of the predefined data items.
- **Manual**: the user manual.

Even if the navigation portlet can run all the visible documents, the portal can be composed by several pages and sections: every portlet addresses a specific document, for a free composition of the informative scene and for an immediate view of the particular business context.

Every user can use the *Released* document according to his role's visibility. When the user owns different roles and the documents have different behaviours, the role for which the document has to be executed is required.

The end user has different freedom degrees of movement and of personalization of the analysis. The modules which allow the greater freedom degree are:

- **QbE**: the user can produce and save in the repository its own interrogations;
- **OLAP**: the user can freely reorient his data model saving his more interesting view.

To be continued.

When executing a document, the end-user can edit some notes directly in the execution page: these notes are correlated to the document functional parameters values, so executions with different parameters values will have different notes. The notes can be disabled/abled by portlet preferences in the execution portlet.

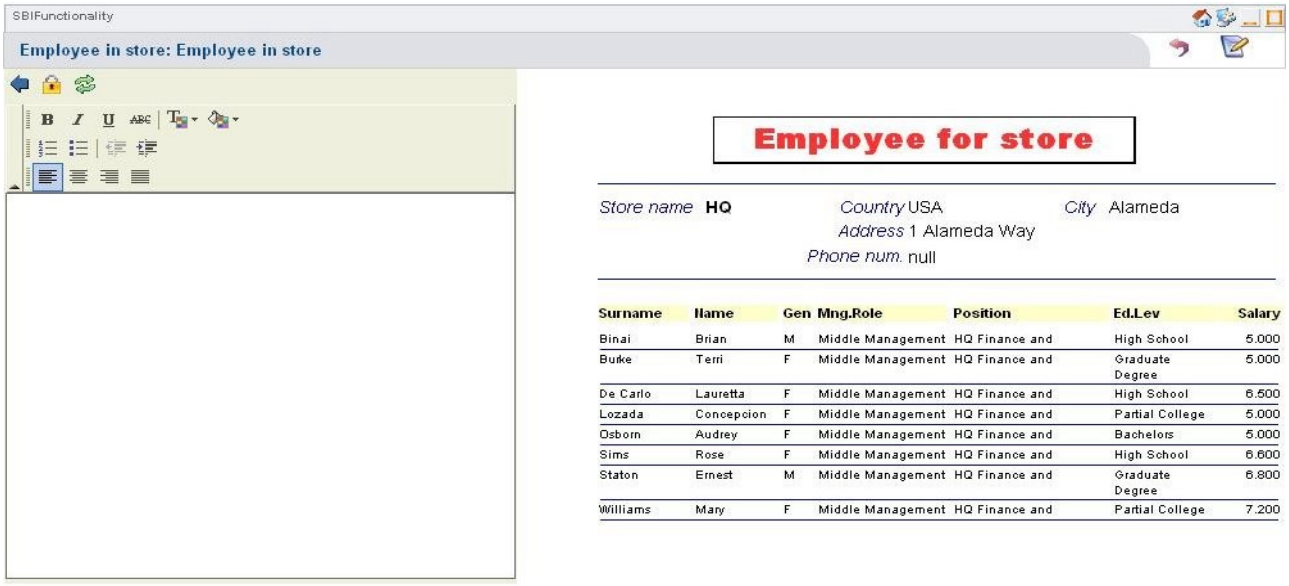

Figura 59 - Notes editor

6.6. Events

In SpagoBI an "event" is the registration of an action that is performed by a user; an event is correlated to some portal roles with a logic that depends on the type of action and it is notified to all the users having at least one of those roles. The images below show a list of a event a event detail.
At the time being only SpagoBIWekaEngine (the engine for data-mining processes integrating Weka) can register events: the engine registers when a process is started and when it is completed (as it is an asynchronous process). These events are notified to the users having at least a role that can execute the data-mining document. In the detail page of the end process event there is a link to the document that will show the process results.

The events management and registration is under a phase of greater integration through.

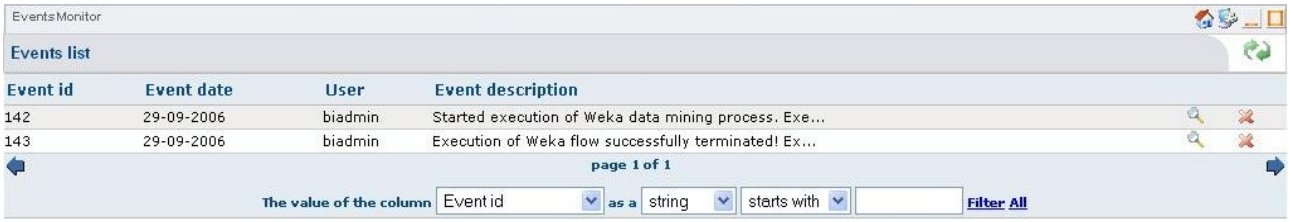

Figura 60 - Events list

Figura 61 - Event detail

7. Getting Started with SpagoBI

This chapter explains, using some examples, the single steps allowing you to enrich the demo portal with new analytical documents, by means of the SpagoBI functionalities.

First of all you have to install ExoTomcat and the SpagoBI DEMO.
How to handle the different categories of analytical documents follows.

This chapter is in revision phase.

7.1. Install ExoTomcat and SpagoBI DEMO

Download ExoTomcat 1.1.3 from :
http://forge.objectweb.org/project/showfiles.php?group_id=151&release_id=1378

Unzip the file exoPortal-tomcat-1.1.3.zip but do NOT run the application, because this would cause the failure of the following SpagoBI installation.

Download SpagoBI Demo 1.8 from:
http://forge.objectweb.org/project/showfiles.php?group_id=204&release_id=1407

Unzip the file.

At the command shell navigate to the directory that contains the jar file obtained and type:

java –jar SpagoBIDemoInstaller.jar

Follow the instructions to complete the installation process. Notice that it will be required to specify the directory where you have previously unzipped ExoTomcat 1.1.3.

Once completed SpagoBI installation, at the command shell it is necessary to navigate to the temp/data/databases directory of Exo-Tomcat and type start.bat for Windows or ./start.sh for Unix: remember to check if you have the required permission for execution.

Then, at the command shell navigate to the /bin directory of Exo-Tomcat and type startup.bat for Windows or ./startup.sh for Unix.
You can read the log file by typing:

tail –f ../logs/catalina.out

7.2. Report

The main steps to manage a report are:
1. Create a Report Template
2. Create Parameters
   a. Create Lists of Value

b.Create Constraints
3.Register the Analytical Document (the built report) into the platform
a.Add Template
b.Assign Parameters
4.Test the Analytical Document
5.Execute the Analytical Document

Notice that JasperReport is the first report engine chosen, but it is not the only one allowed. Similarly iReport is the first chosen interface for JasperReport engine but other solutions also exists and the developer can use what he prefers in order to produce the report template.

7.2.1.Create a Report template

7.2.2.Create a Report template using iReport

1.If you don't have iReport 1.2.5 you can download it from:
http://sourceforge.net/project/showfiles.php?group_id=64348

2.Unzip the downloaded file.

3.Before starting iReport it is necessary to copy the file *hsqldb1_8_0_2.jar*, that can be found in the common/lib directory of Exo-Tomcat, in the /lib directory of iReport. Furthermore delete the *hsqldb1_61.jar* file.

4.Now it's possible to run iReport.

5.In order to create a simple report example, select **New Document** from the **File** menu. Type *SpagoBI_Example* as Report name and click on **OK**.
From the **Datasource** menu select **Connection/Datasources.**
Click on **New** and fill in the following information:
•**Name**: SpagoBI_foodmart
•**JDBC Driver**: org.hsqldb.jdbcDriver
•**JDBC URL**: jdbc:hsqldb:hsql://localhost/foodmart
•**Username**: sa

Select **Save.**

6.Open the **Report query** window from the **Datasource** menu and in the **Report SQL query** text field enter the following example query:

```
select FIRST_NAME, LAST_NAME
from EMPLOYEE e, POSITION p
where p.POSITION_ID = e.POSITION_ID
and p.POSITION_TITLE='$P{EmployeePosition}'
```

This simple query will visualize the First Name and the Last Name of every employee whose position title is equal to the value of the parameter *EmployeePosition*.

7.From the **Project Browser** on the left hand side of the window, expands **Object library**.
Right-click on the Project Browser and click on **Add** and then on **Parameter.**
Type *EmployeePosition* in the **Parameter name** field, selecting the **Is for prompting** check box and then select **OK**.

8.It is now possible to create the report layout.
Select the **"T"** icon from the top toolbar and insert a new text field in the **Detail** area of the report. Double-clicking on the new field and in the **Static Text,** type *First Name*.
Exit the properties dialog box and create another text field positioning in the **Detail** area below the first one.

Double-click on the **Fields** element in the **Project Browser** and drag the FIRST_NAME field next to the corresponding static text field just created. Repeat the same action for the LAST_NAME field.

This report will simply display a list of all employees obtained by the query.

9.Now it is possible to **execute the report** by selecting the corresponding command from the **Build** menu.

Before displaying the final result, the application will ask you to enter the EmployeePosition parameter value. You can type: "President".

7.2.3.Create a Report template using Birt

1.If you don't have Birt 2.0.1 you can download it from:
http://download3.eclipse.org/birt/downloads/
We suggest to download the all-in-one package you find under "Report Designer Full Eclipse Install" section.

2.Before starting eclipse it is suggested (but it is not mandatory) to download the Eclipse Birt plugin you find in
http://forge.objectweb.org/project/showfiles.php?group_id=204
This Eclipse-Birt plugin will permit SpagoBIBirtReportEngine to use a given JDBC connection instead of retrieving one by the ODA framework; in the former case the connection can be set as a document parameter and can be changed at runtime (changing the parameter value) without editing the document template; in the latter case the parameter for the connection to be used will be completely unuseful and the only way to change connection will be editing the document template, changing connection configuration and uploading the modified template.
Unzip the file EclipseBirtPlugins-bin-1.9RC.zip inside the Eclipse home directory;

3.Then start the Foodmart HSQL db and copy the database driver hsqldb1_8_0_2.jar inside the following directory:
ECLIPSE-HOME\plugins\org.eclipse.birt.report.data.oda.jdbc_2.0.1\drivers.

4.Now you can start Eclipse-Birt.

5.In order to create a simple report example, select New Project → Business Intelligence and Reporting Tools → Report Project from the File menu. Then select New Report from the same menu. Type SpagoBI_Example as Report name and click "Finish".

6.From the Data Explorer view rigth click on Data Sources and edit a new Data Source. If you have installed the Eclipse Birt plugin as described at point 2, choose "External (or standard) JDBC Data Source" as data source type. Please consider that this plugin will permit to use only one JDBC connection so you cannot use this plugin for more than one data sources. If you need to use more than one data sources or if you haven't installed the Eclipse Birt plugin, you can configure them as standard "JDBC Data Source" (ODA framework will be actived). Enter "foodmart" as Data Source name and click "Next". Then fill in the following information:

- **JDBC Driver**: org.hsqldb.jdbcDriver (if you don't see this driver from the list go to "Manage drivers" and add the driver for HSQL selecting the jar *hsqldb1_8_0_2.jar* copied at point 3)
- **JDBC URL**: jdbc:hsqldb:hsql://localhost/foodmart
- **Username**: sa

Click "Finish".

7.Then right-click on Data Sets and add a new Data Set with the following information:

- **Data Set name**: Employees
- **Data Source**: Foodmart
- **Data Set Type**: SQL Select Query

8.In the Edit Data Set window, inside **Query** view, insert the following example query:

```
select FIRST_NAME, LAST_NAME
from EMPLOYEE e, POSITION p
where p.POSITION_ID = e.POSITION_ID
and p.POSITION_TITLE= ?
```

Then go to **Parameters** view and add this parameter:

- **name**: EmployeePosPar
- **Data Type**: String
- **Direction**: input
- **Default Value**: 'Store Manager'

This simple query will visualize the First Name and the Last Name of every employee whose position title is equal to the value of the parameter *EmployeePosPar*.

9.Using drag-and-drop from Palette view to report layout, add a table with 2 columns and 1 detail row: using drag-and-drop from the Employees data set put FIRST_NAME on the detail row of the first column and LAST_NAME on the detail row of the second column.

10.In Data Explorer View right-click on "Report Parameters" and add a new Parameter:

- **name**: EmployeePosition
- **Data Type**: String
- **Display type**: Text box
- **Default Value**: Store Manager

Then, select the table you added at point 9., click on "Binding" in the table Property Editor and substitute the default value of the data set parameter EmployeePosPar in the Expression Builder window with:
params["EmployeePosition"]

11.Now it is possible to see the report preview clicking on "Preview" button; to change the parameter value click on "Show Report Parameters" button.

7.2.4.Create a Parameter

Connect to the home page of SpagoBI portal (http://localhost:8080/sbiportal) and log on using both "bidev" as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page.

To **create a new Parameter** the following steps are required:

1.Predefined List of Value (LOV);
2.Predefined Values Constraints;
3.Parameters Management.

7.2.5.Predefined List of Value (LOV)

1.The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools* .

2.Click on the Insert icon to add the new desired element.

3.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: Report - LOV QUERY
- **Name**: Report - LOV QUERY
- **Description**:
- **Input Type**: Query statement

Then, in the *Wizard Query* fill in as follows:

- **Connection Name:** Foodmart Data Warehouse
- **Query Definition:** select POSITION_TITLE from POSITION
  (then click on "Update fields from query")
- **Visible Columns:** POSITION_TITLE
- **Value Column:** POSITION_TITLE

4.Once completed the data entry, click on the Save icon saving the information and exit to the *Predefined List of Values* page.

5.Now click again on the Insert icon to create a second LOV.

6.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: Report - LOV FIX_LOV
- **Name**: Report - LOV FIX_LOV
- **Description**:
- **Input Type**: Fixed list of values

Then, in the *Wizard Fix Lov* add the following pairs:

- **New item name**: HQ Information System
  New item value: HQ Information System
  (click on the ADD icon)
- **New item name**: HQ Marketing
  New item value: HQ Marketing
  (click on the ADD icon)
- **New item name**: HQ Human Resources
  New item value: HQ Human Resources
  (click on the ADD icon)
- **New item name**: HQ Finance and Accounting
  New item value: HQ Finance and Accounting
  (click on the ADD icon)

Notice that usually *Name* is the field that allows the comprehension of the *Value* field.

7.When the data entry is completed, click on the SAVE icon saving the information and exit to the *Developer Tools* page.

7.2.6.PREDEFINED VALUES CONSTRAINTS

1.The *Predefined Values Constraints* page can be accessed by selecting the corresponding link from the *Developer tools*. It is divided into two parts: on the top side a list of *Predefined Constraints* is displayed; on the bottom the *Configurable Constraints*.

2.Click on the INSERT icon to access the *Constraint Details* page and create a new constraint.

3.Insert the following information:

- **Label**: Report - Constraint
- **Name**: Report - Constraint
- **Desription:**

4.Select *MAXLENGHT* as *Check Type* and type *23* in the corresponding text field.

5.When completed, click on the SAVE icon saving the information and exit to the previous page; then select the BACK icon to go to the *Developer Tools* page .

7.2.7.PARAMETERS MANAGEMENT

When the required LOV and CHECK are created, a new *Parameter* can be created too.

1.Enter the Parameter List page by selecting Parameters Management from the Developer Tools page.

2.Click on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label**: Report - Parameter
- **Name**: Report - Parameter
- **Description**:
- **Type**: String

Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.
Enter the following information:

- **Label :** Report - Use Mode 1
- **Name:** Report - Use Mode 1
- **Description.**

In the *Role Association* table, select */spagobi/admin* and */spagobi/dev*.
The Administrator or Developer executing a document associated to this parameter will use this specific *Use Mode*.

Then click on LOV LOOKUP icon; in the *Lov lookup list* page now displayed select *Report - LOV QUERY* from the table listing all *Predefined List of Values* (the filter will help you to find it soon).

From the *Predefined Values Constraints* table don't select any constraints.

6.Now click on the SAVE icon.

It can be useful to add another Use Mode in order to understand the roles management performed by *Parameters*.

7.Click on *New…* tab and a new (empty) *Parameter Use Mode Details* section will appear.

8.Fill in the following information:

- **Label:** Report - Use Mode 2
- **Name:** Report - Use Mode 2
- **Description**

Then click on LOV LOOKUP icon; in the *Lov lookup list* page now displayed select *Report - LOV FIX_LOV* from the table.

Select the *Report - Constraint* from the *Predefined Values Constraints* list.

Notice that in the *Role Association* table the */spagobi/admin* and */spagobi/dev* cannot be selected. In fact, each role can be matched at most to one *Use Mode*.
Check the */spagobi/biuser.*

9.Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The *Report - Parameter* will be now displayed in the list. Notice that the *Number of Use Modes* should be 2.

Then click again on the BACK icon to reach the *Developer Tools*.

7.2.8.REGISTER THE ANALYTICAL DOCUMENT (THE BUILT REPORT) INTO THE PLATFORM

1.From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.

2.To create a new *Analytical Document* it is necessary to select the INSERT icon.

3.In new *Document Details* page you have to fill in the following information:

- **Label**: Report - Document
- **Name**: Report - Document
- **Description**
- **Type**: Report
- **Engine**: "Jasper Report Dev" if you created the template using iReport, or "Birt Engine" if you created the template with Birt
- **Template**: click on the *browse* button to select the report template created in the Create a Report template using iReport or in the Create a Report template using Birt paragraph.

4.Moreover, it is necessary to indicate which is the parent folder of the document by selecting the check box corresponding to *Static Reporting*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.

5.To save and exit from this page click on the SAVE AND RETURN icon.

6.The *Development Objects List* will be updated with a new row containing the document just created.

7.Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.

8.In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the '*NEW …*' tab: insert the following information:

- **Title**: Report - Doc Param
- **URL Name**: EmployeePosition

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select *Report - Parameter* from the table.

Notice that the URL Name must match the name of the parameter created in the report template example in the Create a Report template or Create a Report template using Birt paragraph.

9.When this operation is completed, save information by clicking on the SAVE icon.

10.It is possible to create a standard parameter in order to choose the output format of the document. Click on the '*NEW …*' tab and add the following information:

- **Title**: Choose output format
- **Parameters**: Output Type
- **URL Name**: param_output_format

Notice that the Parameter is already present in the SpagoBI DEMO. Moreover the same parameter is used by many of the existing Analytical Document listed in this DEMO. This is an example of how to use the same resource for different documents.

Furthermore, notice that the URL Name refers to a predefined parameter for the *Jasper Report* and *Birt* engines.

11. When this operation is completed, save information by clicking on the SAVE AND RETURN icon. The new parameter will be displayed in the list.

12. Go back to the *Development Object List* by clicking on the BACK icon.

Now, all the parameters have been correctly created and configured on the *Analytical Document*. So you can execute it.

13. Click on the EXECUTE icon corresponding to the new *Report - Document*.

14. In the new page you will have to enter a value for the *Report - Doc Param*.

15. Click on the DETAILS icon and choose one of the predefined values in the list by clicking on the corresponding DETAILS icon. Select *HQ Information System.*

16. Click on the EXECUTE icon to execute the document.

17. Now it is possible to Update State by clicking on the corresponding icon at the bottom of the window. Notice that when the *Document Example* is updated, it will not be listed in the *Development Object List* anymore.

18. Logout

7.2.9. TEST THE ANALYTICAL DOCUMENT

Work in progress.

7.2.10. EXECUTE THE ANALYTICAL DOCUMENT

Work in progress.

7.3. OLAP ANALISYS

This following example is designed to quick introduce new users to create a new OLAP in SpagoBI.

The main steps to manage a report are:
1. Create a Template
2. Create Parameters
   a. Create Lists of Value
   b. Create Constraints
3. Register the Analytical Document (the built OLAP) into the platform

a.Add Template
b.Assign Parameters
4.Test the Analytical Document
5.Execute the Analytical Document

7.3.1.Create a template

In order to create a proper template for an OLAP document, it is only necessary to realize an XML file containing the following elements:

- <olap> root element;
- <cube> the *reference* attribute of this element identifies the XML file that describes the datamart cube using the Mondrian syntax.
- <MDXquery> the text of this element is a query executed on the datamart, written in the MDX syntax.
- <parameter> this element contained in the <MDXquery> identifies a parameter for the query. The *name* attribute refers to the parameter name contained in the query, while the *as* attribute identifies the alias that will be used by the SpagoBI document.

For this example it is necessary to create an XML file containing the following text:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<olap>
  <cube reference='/WEB-INF/queries/FoodMart.xml' />
        <MDXquery>
         select
         {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} on
columns,
          {Parameter("ProductMember", [Product], [Product].[All Products].[Food], "wat
willste?").children} ON rows
         from Sales where ([Time].[1997])
   <parameter name='prdCd' as='ProductMember' />
        </MDXquery>
</olap>
```

Save the file as *productSales.olap*.

Please note that the example above uses a parameter with the proper MDX sintax. You can also insert parameters in a MDX qyery using SpagoBI sintax *${parameter_name}*: in this case the parameter value will literally substitute the parameter declaration before query execution. The example above could be:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<olap>
  <cube reference='/WEB-INF/queries/FoodMart.xml' />
        <MDXquery>
         select
         {[Measures].[Unit Sales], [Measures].[Store Cost], [Measures].[Store Sales]} on
columns,
         ${ProductMember} ON rows
         from Sales where ([Time].[1997])
   <parameter name='prdCd' as='ProductMember' />
```

```
        </MDXquery>
    </olap>
```

Using the SpagoBI sintax you can insert more than one parameter in the same dimension in a very easy way (example: [Time].[${year}].[${month}]).

7.3.2.Create Parameters

Connect to the home page of SpagoBI portal (http://localhost:8080/portal) and log on using "bidev" both as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page .

To **create a new Parameter** the following steps are required:
1.Predefined List of Value (LOV);
2.Predefined Values Constraints;
3.Parameters Management.

7.3.3.Predefined List of Value (LOV)

1.The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.

2.Click on the Insert icon to add the new desired element.

3.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: OLAP - LOV FIX_LOV 1
- **Name**: OLAP - LOV FIX_LOV 1
- **Description**
- **Input Type**: Fixed list of values
- **New item name**: Non Consumable
  New item value: [Product].[All Products].[Non-Consumable]
  (click on the ADD icon)
- **New item name**: Food
  New item value: [Product].[All Products].[Food]
  (click on the ADD icon)
- **New item name**: Drink
  New item value: [Product].[All Products].[Drink]
  (click on the ADD icon)

4.When the data entry is completed, click on the Save icon to save the information and exit to the *Predefined List of Values* page.

5.Now click again on the Insert icon to create a second LOV.

6.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: OLAP - LOV FIX_LOV 2
- **Name**: OLAP - LOV FIX_LOV 2
- **Description**:
- **Input Type**: Fixed list of values
- **New item name**: Non Consumable
  New item value: [Product].[All Products].[Non-Consumable]
  (click on the ADD icon)
- **New item name**: Food
  New item value: [Product].[All Products].[Food]
  (click on the ADD icon)

7.When the data entry is completed, click on the SAVE icon to save the information and exit to the *Developer Tools* page.

7.3.4.PREDEFINED VALUES CONSTRAINTS

1.The *Predefined Values Constraints* page can be accessed by selecting the corresponding link from the *Developer tools*. It is divided into two parts: on the top side a list of *Configurable Constraints* is displayed; on the bottom the *Predefined Constraints*.

2.Click on the INSERT icon to access the *Constraint Details* page and create a new constraint.

3.Insert the following information:

- **Label**: OLAP - Constraint
- **Name**: OLAP - Constraint
- **Desription**

4.Select *MAXLENGHT* as *Check Type* and type *35* in the corresponding text field.

5.When completed, click on the SAVE icon to save the information and exit to the previous page. Then select the BACK icon to go to the *Developer Tools* page.

7.3.5.PARAMETERS MANAGEMENT

When the required LOV and constraints are created, a new *Parameter* can be created too.

1.Enter the *Parameter List* page by selecting *Parameters Management* from the *Developer Tools* page.

2.Click on the INSERT icon and open the *Parameter Details* page.

3.Insert the following information:

- **Label**: OLAP - Parameter
- **Name**: OLAP - Parameter
- **Description**
- **Type**: String

4.Click on the SAVE icon. An empty *Parameter Use Mode Details* section will be displayed. Enter the following information:

- **Label :** OLAP - Use Mode 1
- **Name:** OLAP - Use Mode 1
- **Description**

In the *Role Association* table, select */spagobi/admin* and */spagobi/dev*.
The Administrator or the Developer executing a document associated to this parameter, will use this specific *Use Mode*.

Then select *OLAP - LOV FIX_LOV 1* from the lookup page listing all Predefined List of Values.

From the *Predefined Values Constraints* table don't select any constraints.

5.Now click on the SAVE icon.

It can be useful to add another Use Mode in order to understand the roles management performed by *Parameters*.

6.Click on the '*NEW …*' tab and add the following information:

- **Label:** OLAP - Use Mode 2
- **Name:** OLAP - Use Mode 2
- **Description**

Then select *OLAP - LOV FIX_LOV 2* from the lookup page listing all Predefined List of Values.

This time select the *OLAP - Constraint* from the *Predefined Values Constraints* list.

Notice that in the *Role Association* table the */spagobi/admin* and */spagobi/dev* cannot be selected. In fact each role can be matched at most to one *Use Mode*.
This time check the */spagobi/biuser.*

7.Click on the SAVE AND RETURN icon to go back to the *Parameters List*. The *OLAP Parameter Example* will be now displayed in the list. Notice that the *Number of Use Modes* should be 2.

8.Then click again on the BACK icon to reach the *Developer Tools*.

7.3.6.REGISTER THE ANALYTICAL DOCUMENT (THE BUILT OLAP) INTO THE PLATFORM

1.From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.

2.Select the icon to create a new *Analytical Document*.

3.In new *Document Details* page you will be required to fill in the following information:

- **Label**: OLAP - Document
- **Name**: OLAP - Document

- **Description**:
- **Type**: On-line analytical processing
- **Engine**: Jpivot-Mondrian Dev
- **Template**: click on the *browse* button to select the template created in the paragraph.

4.Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dimensional Analysis*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.

To save and exit from this page click on the SAVE AND RETURN icon.

The *Development Objects List* will be updated with a new row containing the document just created.

Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.

In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the '*NEW …*' tab: insert the following information:

- **Title**: OLAP - Doc Param
- **URL Name**: prdCd

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select *OLAP – Parameter* from the table.

Notice that the URL Name must match the alias (as attribute) of the parameter created in the template example created in the paragraph.

Save and go back to the *Development Object List* by clicking on the SAVE AND RETURN icon.

Now that all the parameters have been correctly created and configured on the *Analytical Document*, you can execute it.

Click on the icon EXECUTE which corresponds to the new *OLAP - Document*.

In the new page you will be required to enter a value for the *OLAP - Doc Param*.

Select *Food* from the combo-box.

Click on the EXECUTE icon to execute the document.

Now it is possible to Update State by clicking on the corresponding icon at the bottom of the window. Notice that once updated the *OLAP - Document* will not be listed in the *Development Object List* anymore.

Logout

7.3.7. Test the Analytical Document

Work in progress.

7.3.8. Execute the Analytical Document

Work in progress.

7.3.9. Dynamic Template Creation

Since SpagoBI version 1.9.2, it is possible to build the template of an OLAP document dynamically, in alternative of what you read in chapter 7.3.1.
Let's see an example. Configure a new document:

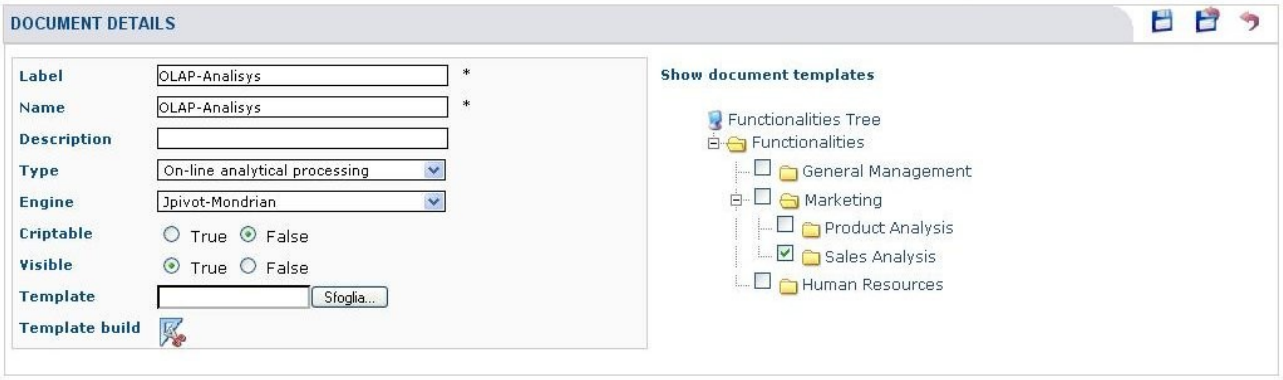

Figura 62 - New OLAP document

Then click on the Save button without inserting any template file. Then click on the template build button.
Now you have to specify the connection to be used and the schema definition available (available connections and schema are specified in exo-home/webapps/SpagoBIJpivotEngine/ WEB-INF/classes/engine-config.xml).

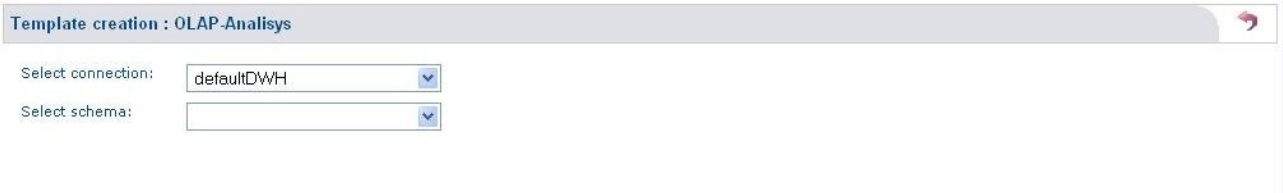

Figura 63 - Connection and schema selection

Then you have to specify the analisys cube.

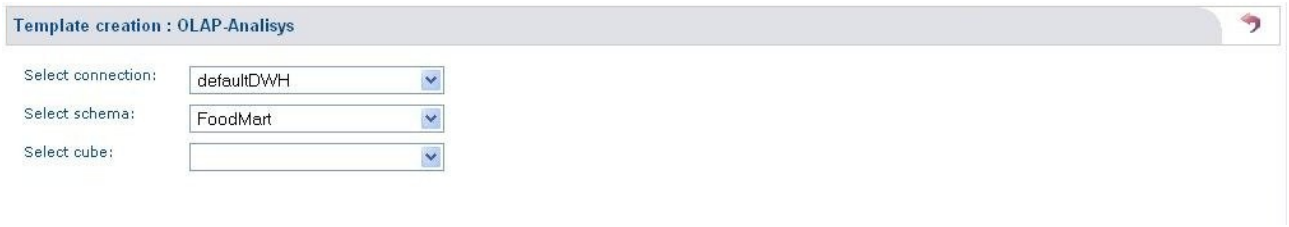

Figura 64 - Cube selection

After you chose the cube, you can use the olap cube navigator to specify the dimensions and measures to be displayed.

Figura 65 - Cube navigation

Then you can save the template (without parameters): click on the save template button and fill the form with the template name, then click on the "Save" button.

Figura 66 - Saving template without parameters

If you want to insert also one or more parameter click on the "Add parameters" button: you will see the MDX query you have obtained using the cube navigator.

Figura 67 - Add parameters page

You can modify the query in order to insert a parameter using both the Mondrian sintax or the SpagoBI sintax (in the example below you see a SpagoBI sintax usage), then you have to insert the parameter name and url name in the form.

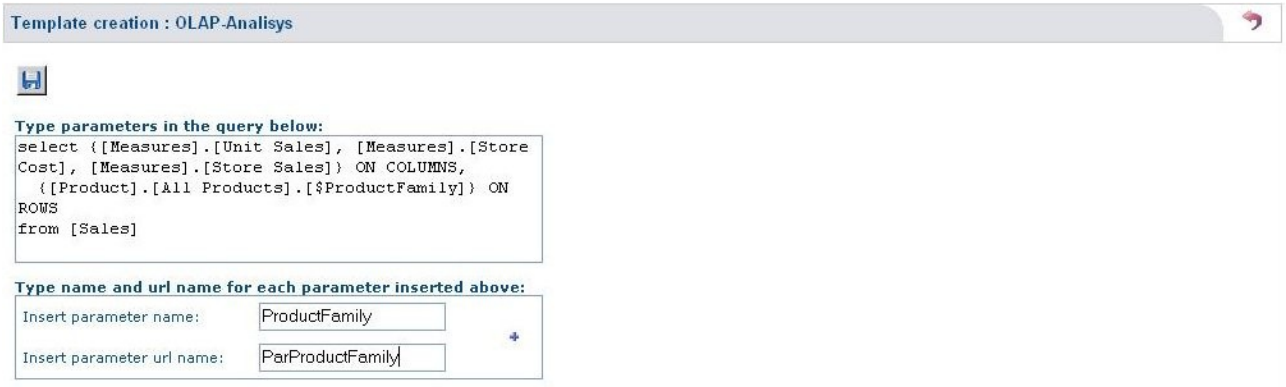

Figura 68 - Parameter definition

Then to click on the "Add parameter" button.

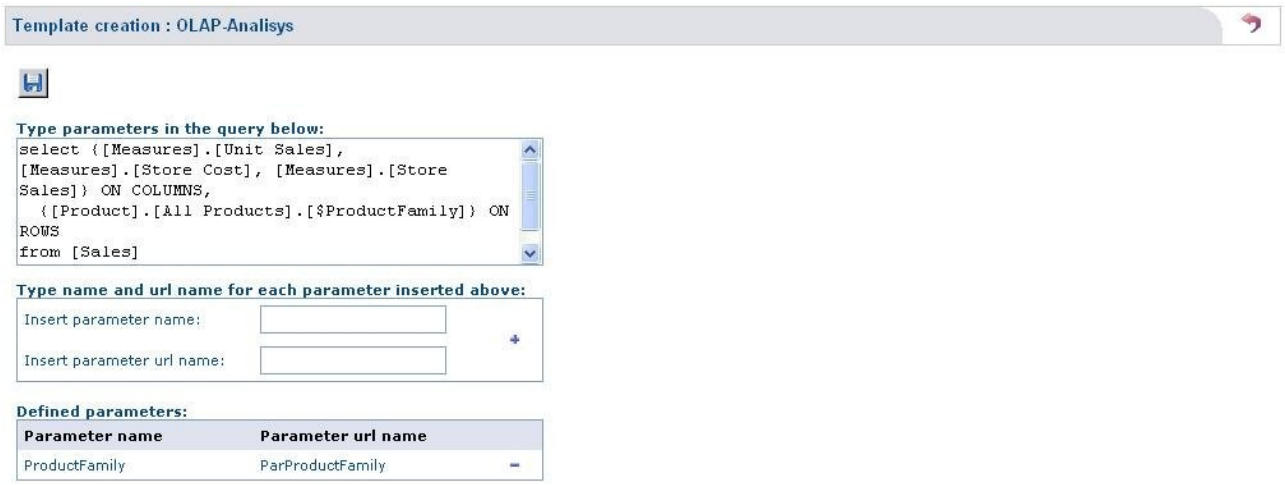

Figura 69 - Parameter inserted

Then you can save the new template by clicking on the "Save template" button, filling the form with the name of the template and then clicking the "Save" button.
At last you have to configure the parameter as a document parameter: supposing you have previously defined a fixed list of values with values "Food", "Drink" and "Non-Consumable" (remember that we used the SpagoBI parameter sintax) and associated this lov to a parameter named "Product family", configure a document parameter as in the figure below (note that the parameter url name must be equal to the url name you put during the template creation procedure).

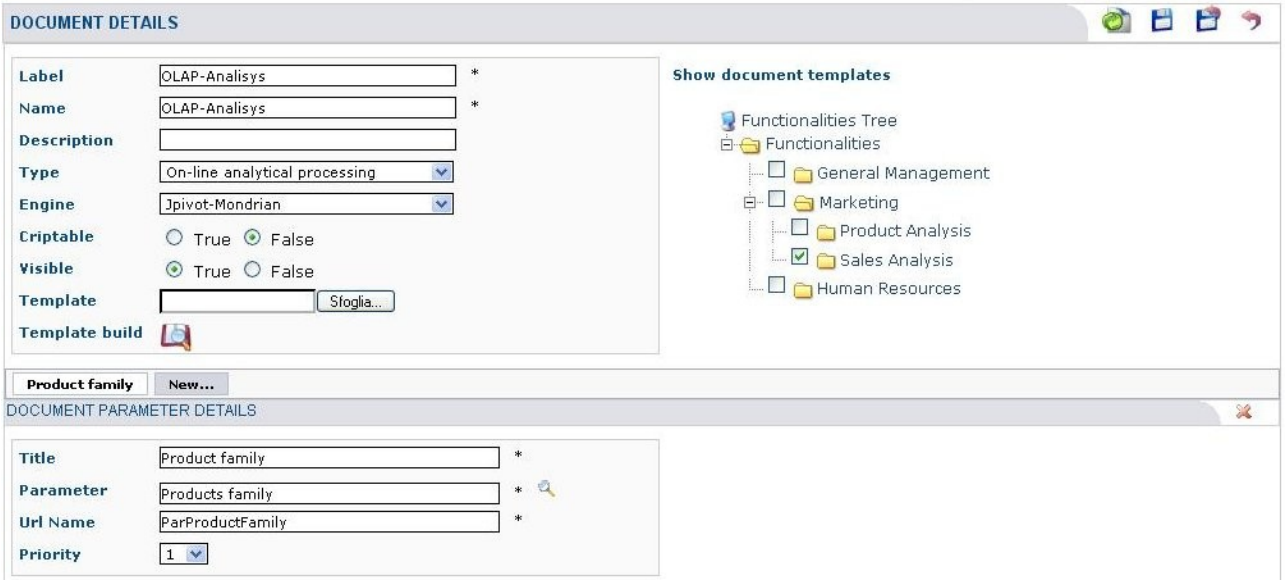

Figura 70 - Document parameter definition

Note that you can go into the template build page to modify the query (you will have to specify the connection to be used, while schema and cube will remain the same). If you inserted parameters with SpagoBI sintax, the system will execute the last Mondrian query (the query

obtained before adding parameters); use the Jpivot interface to modify the query and then re-define the parameters as seen before.

7.4. Dashboard

This following example is designed to quick introduce new users to create a new DASHBOARD in SpagoBI starting from a SWF movie already created with OpenLazslo.

In this document we will not explain how to build a SWF movie with OpenLazslo; we invite you to visit the OpenLazslo home page www.openlaszlo.org for more information.

SpagoBI is released with some SWF movies; you can find them in the folder *webapps/spagobi/dashboards* inside Exo-Tomcat main directory. For this example we will consider *rot.lzx.swf*. For the source code and documentation about SpagoBI movies download the package SpagoBI Components from http://forge.objectweb.org/project/showfiles.php?group_id=204&release_id=1407.

A Dashboard element must be fed with the values to be displayed: these values are supplied by a predefined List of Value.

The main steps to manage a dashboard are:
1. Create the Lists of Value for movie feeding
2. Create a Template for movie configuration
3. Register the Analytical Document (the built DASHBOARD) into the platform
    a. Add Template
4. Test the Analytical Document
5. Execute the Analytical Document

7.4.1. Create a list of value

1. The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools*.

2. Click on the Insert icon to add the new desired element.

3. In the *Predefined List of Values Details* page fill in the following information:

- **Label**: DASHBOARD - SCRIPT
- **Name**: DASHBOARD - SCRIPT
- **Description**
- **Input Type**: Script
- **Script**:

```
StringBuffer buf = new StringBuffer();
buf.append('<rows>');
buf.append('<row value=\'');
Random rand = new Random();
float f = 80 + ((rand.nextFloat() * 100) % 15);
buf.append(f);
buf.append('\' />');
```

```
    buf.append('</rows>')
    return buf;
```

•**Output**: Single Value

The above script will only return a random value.

4.When the data entry is completed, click on the SAVE icon to save the information and exit to the *Predefined List of Values* page.

7.4.2.CREATE A TEMPLATE

In order to create a proper template for a DASHBOARD document, it is only necessary to realize an XML file containing the movie configuration: create an XML with the following content:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
        <DASHBOARD movie="dashboards/rot.lzx.swf" displayTitleBar="true">
                <DIMENSION width="240" height="150" />
                <CONF>
                        <PARAMETER name="minValue" value="0" />
                        <PARAMETER name="maxValue" value="200" />
                        <PARAMETER name="lowValue" value="50" />
                        <PARAMETER name="highValue" value="120" />
                </CONF>
                <DATA url="DashboardService">
                        <PARAMETER name="dataname" value="DASHBOARD - SCRIPT" />
                 </DATA>
</DASHBOARD>
```

Save the file as DASH_EXAMPLE.xml.

Notice the following: the *movie* attribute of the *DASHBOARD* tag refers to the SWF file in SpagoBI context; *DIMENSION* and *CONF* tags refers to the movie configuration; DATA tag defines the Servlet and the List of Value for the movie values feeding.

7.4.3.REGISTER THE ANALYTICAL DOCUMENT (THE BUILT DASHBOARD) INTO THE PLATFORM

1.From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.

2.Select the icon to create a new *Analytical Document*.

3.In new *Document Details* page you will be required to fill in the following information:

•**Label**: DASHBOARD - Document

- **Name**: DASHBOARD - Document
- **Description**:
- **Type**: Dashboard
- **Template**: click on the *browse* button to select the template created in the previews paragraph.

5.Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dashboard*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page.

6.To save and exit from this page click on the SAVE AND RETURN icon.

Now that the document has been correctly created and configured on the *Analytical Document*, you can execute it.

7.Click on the icon EXECUTE which corresponds to the new *DASHBOARD - Document*.

8.Now it is possible to Update State by clicking on the corresponding icon at the top of the window. Notice that once updated the *DASHBOARD - Document* will not be listed in the *Development Object List* anymore.

9.Logout

7.4.4.TEST THE ANALYTICAL DOCUMENT

Work in progress.

7.4.5.EXECUTE THE ANALYTICAL DOCUMENT

Work in progress.

7.5.DATA MINING

The main steps to menage a data mining process are:

1.Design a data mining process using Weka KnowledgeFlow
2.Save the process flow in the kflm format
3.Parametrize the process flow

4.Create Parameters
   a. Create Lists of Value
   b. Create Constraints
5.Register the Analytical Document (the built data mining process) into the platform
   a. Add Template
   b. Assign Parameters
6.Test the Analytical Document
7.Execute the Analytical Document

7.5.1.Create a Data mining process using Weka KnowledgeFlow

1. If you don't have yet Weka-3-4 you can download it from:

   http://www.cs.waikato.ac.nz/ml/weka/

2. Unzip the downloaded file and install the program
3. Add your JDBC driver libbrary to the classpath or simply copy it into the weka installation directory
4. Run weka
5. In order to create a simple data mining process example, select **KnowledgeFlow** button from the **Weka GUI Chooser** window.

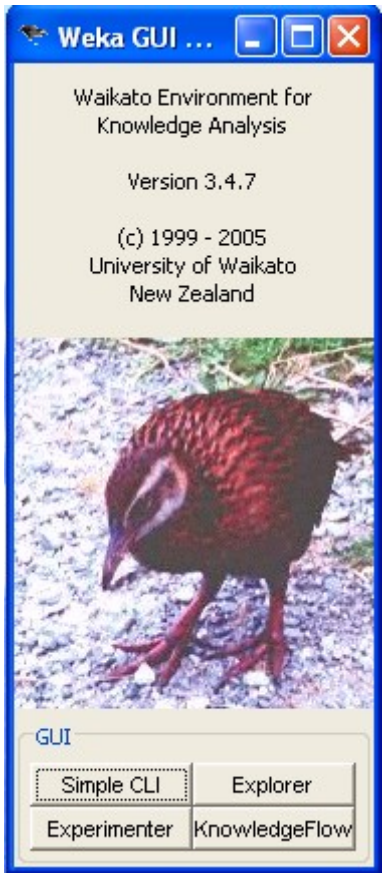

Figure 71 - Weka GUI Chooser window

6. Choose **Database Loader** component from **DataSource Tab** and add it to the flow (for the moment the SpagoBI Weka Engine supports only this kind of DataSource).
7. Right click on Database Loader component and choose **configure option** from the pop-up menu.
8. Set up the query parameter as follow ...

```
SELECT * FROM cust_to_dept
```

   The table `cust_to_dept` have an instance for each customer and a column for each store department recording the total amount of money spent by the specified customer in the last year in that department (i.e. `[cust_id, dep1, dep2, ..., depN]`). Others parameters in this configuration window can be ignored because the connection to the database is setted by the engine according to its configuration settings
9. Choose **Database Saver** component from **DataSink Tab** and add it to the flow (for the moment the SpagoBI Weka Engine supports only this kind of DataSink).
10. Right click on Database Saver component and choose **configure option** from the pop-up menu.
11. Set up the **Table Name parameter** as follow ...

```
cust_to_dept_clustered
```

This table is used to store the outcomes of the mining process. It associates each customer to a cluster (i.e `[cust_id, cluster]`). Others parameters in this configuration window can be ignored because the connection to the database is setted by the engine according to its configuration settings

12. Now you can add to the flow a mining algorithm. Choose, for example, **Add Cluster** component from **Filters Tab**. This component adds a new nominal attribute representing the cluster assigned to each instance of the input dataset by the specified clustering algorithm.
13. Right click on Add Cluster component and choose **configure option** from the pop-up menu.
14. Choose the clustering algorithm to be used by the Add Cluster component. Than choose fields of input instances to ignore during clustering process. In this example we choose Simple-KMeans as clustering algorithms and we choose to ignore the first filed (i.e. the customer id).
15. Right click on DatabaseLoader component and choose **dataset option** from the pop-up menu in order to connect this component to the Add Cluster component. The dataset connection is just a pipe that push the source component output to the destination component input. In this case we pass the whole dataset read from the database by the DatabaseLoader component to the Add Cluster component. **Add Cluster** component process the input datasource ignoring the first field and append to each instance a cluster attribute.
16. Because the output dataset of the Add Cluster component (i.e. `[cust_id, dep1, dep2, ..., depN, cluster]`) doesn't fit with our destination table `cust_to_dept_clustered` (i.e `[cust_id, cluster]`) we can't connect directly the Add Cluster component to the DatabaseSaver component. We have to add **Remove** component between these two component in order to filter out all the unnecessary fields. Choose the **Remove** component from **Filters Tab** and add it to the process. Configure it in order to filter out all the fields in input dataset except the first (customer id) end the last (cluster). Connect the Add Cluster component to the Remove component and than the Remove component to the Database Saver component.
17. Save your newly generated data mining process to a file in *kflm* format.

7.5.2. Parametrize process flow

It is possible to parametrize the mining process just using parameters as value of some flow component configuration properties. A parameter is just a string enclosed in the special characters $P{ and }. The enclosed string is the name given to the parameter.

All added parameters must than be associated with SpagoBI parameters when the document is inserted into the system in order to execute it.

Weka doesn't support parameters. For this reason it isn't always possible to add a parameter directly from the GUI Knowledge Flow Editor. This happen because Weka applies constraints to some properties' value format. Using parameters as value of that properties violates the applied constraints and cause errors (usually default values are used instead the inserted ones). In these cases you have to type your parameter directly inside the generated *kfml* file. For example if you want to parametrize the clustering algorithm used in the previously defined flow you can do that directly from the GUI interface because Weka is very severe about the format of this property. So you have to choose an algorithm, save the flow, open the *kfml* file

with a text editor, look for the name of the algorithm inserted from the GUI interface and replace it with a parameter. Searching for SimpleKMeans in the flow defined above we find this line...

```
<object class="java.lang.String" name="1">weka.clusterers.SimpleKMeans -N 15 -S 10</object>
```

that means that we are going to use SimpleKMeans as clustering algorithm to produce 15 clusters using 10 as the initial seed. It is to possible now to replace this line with the following parametrized one ...

```
<object class="java.lang.String" name="1">$P{clusterer} -N $P{clusterNum} -S 10</object>
```

Play attention when you add parameters to the *kfml* file by hand because than this file cannot be reloaded correctly into the GUI Knowledge Flow Editor. If you want to modify it by hand be sure to have make a copy of the original file before.

7.5.3. Create a Parameter

Connect to the home page of SpagoBI portal (http://localhost:8080/portal) and log on using both "bidev" as username and password. This user is a *Developer* for the SpagoBI Demo and therefore you will access the *Developer Tools* page.

To **create a new Parameter** the following steps are required:

1. Predefined List of Value (LOV);
2. Predefined Values Constraints;
3. Parameters Management.

7.5.4. Predefined List of Value (LOV)

1.The *Predefined List of Values* page can be accessed by selecting the corresponding link from the *Developer tools* .

2.Click on the Insert icon to add the new desired element.

3.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: HSQL_FM_CONN
- **Name**: Connection to foodmart on Hsql db
- **Description**: Connection to foodmart on Hsql db
- **Input Type**: Script

Then, in the *Wizard Script* fill in as follows:

- **Script:** 'hsqlfoodmart'
- **Output:** Single Value

where *hsqlfoodmart* is the name given to the hsql connection in the engine configuration file(see engine configuration)

4.Once completed the data entry, click on the Save icon and exit to the *Predefined List of Values* page.

5.Now click again on the INSERT icon to create a second LOV.

6.In the *Predefined List of Values Details* page fill in the following information:

- **Label**: DM_CLUSTERER_VAL
- **Name**: Clustering Algorithms
- **Description**: Clustering Algorithms
- **Input Type**: Fixed list of values

Then, in the *Wizard Fix Lov* add the following pairs:

- **New item name**: Simple K-Means
  New item value: weka.clusterers.SimpleKMeans
  (click on the ADD icon)
- **New item name**: Farthest First
  New item value: weka.clusterers.FarthestFirst

Notice that usually *Name* is the field that allows the comprehension of the *Value* field.

7.When the data entry is completed, click on the Save icon saving the information and exit to the Developer Tools page.

7.5.5.PREDEFINED VALUES CONSTRAINTS

Add constraints if you need them. To add constraints follow the same steps described in session 5.2.2.2.

7.5.6.PARAMETERS MANAGEMENT

When the required LOV and CHECK are created, a new *Parameter* can be created too.

1. Enter the Parameter List page by selecting Parameters Management from the Developer Tools page.

2. Click on the INSERT icon and open the *Parameter Details* page. Insert the following information:
   - **Label**: HSQL_FM_CONN_PAR
   - **Name**: Connection to foodmart on Hsql db

- **Description**: Connection to foodmart on Hsql db
- **Type**: String

Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT
- **Name:** Default Modality
- **Description:** Default Modality
- **Lov:** select and lookup for HSQL_FM_CONN LOV

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

4. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The HSQL_FM_CONN_PAR will be now displayed in the list.

5. Now click again on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label: DM_CLUSTERER_PAR**
- **Name: Clustering Algorithms**
- **Description: Clustering Algorithms**
- **Type: String**

6.Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT
- **Name:** Default Modality
- **Description:** Default Modality
- **Lov:** select and lookup for DM_CLUSTERER_VAL LOV

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

7. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The DM_CLUSTERER_PAR will be now displayed in the list.

8. Now click again on the INSERT icon and open the *Parameter Details* page. Insert the following information:

- **Label: DM_CLUSTERS_NUM_PAR**
- **Name: Number of clusters**
- **Description: Number of clusters**
- **Type: String**

9. Click on the SAVE icon and then a new (empty) *Parameter Use Mode Details* section will be displayed on the bottom part of the page.

Enter the following information:

- **Label :** DEFAULT

- **Name:** Default Modality
- **Description:** Default Modality
- **Manula Input:** select

In the *Role Association* table, select all roles in order to associate this *Use Mode* to all users of the system during document execution.

10. Click on the SAVE AND RETURN icon to go back to the *Parameter List*. The DM_CLUSTERS_NUM_PAR will be now displayed in the list.

7.5.7. REGISTER THE ANALYTICAL DOCUMENT (THE BUILT DATA-MINING FLOW) INTO THE PLATFORM

1. From the *Developer Tools* page, select the *Documents Configuration* in order to display the *Development Object List*.

2. Select the icon to create a new *Analytical Document*.

3. In new *Document Details* page you will be required to fill in the following information:

   - **Label**: USER_SEG_DMP
   - **Name**: User segmentation process
   - **Description**: User segmentation process
   - **Type**: Data-Mining Model
   - **Engine**: Weka Dev
   - **Template**: click on the *browse* button to select the template created in the previous paragraph.

4. Moreover, you have to indicate the parent folder of the document selecting the check box corresponding to *Dimensional Analysis*, located as child of *Analytical Area* in the *Functionality Tree* on the right hand side of the page

5.To save and exit from this page click on the SAVE AND RETURN icon.

6. The *Development Objects List* will be updated with a new row containing the document just created.

7. Now you can access the *Document Details* page simply by clicking on the DETAILS icon on the row of the new document. This page will list the general detailed information of the document. On the right side of the page a new table listing the just added template is displayed.

8. In order to set *Document Parameters* it is necessary to fill the *DOCUMENT PARAMETER DETAILS* form below the '*NEW …*' tab: insert the following informations:

   - **Title**: Connection
   - **URL Name**: connectionName

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select HSQL_FM_CONN_PAR from the table.

9. Select again the *'NEW …'* tab and insert the following informations:

- **Title**: Clusterer
- **URL Name**: clusterer

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select DM_CLUSTERER_PAR from the table.

Notice that the URL Name must match the name of the parameter created in the flow example created in the previous paragraph.

10. Select again the *'NEW …'* tab and insert the following informations:

- **Title**: ClusterNumber
- **URL Name**: clusterNum

Then click on PARAMETERS LIST icon; in the *PARAMETERS LIST* page now displayed select DM_CLUSTERS_NUM_PAR from the table.

Notice that the URL Name must match the name of the parameter created in the flow example created in the previous paragraph.

11. Save and go back to the *Development Object List* by clicking on the SAVE AND RETURN icon.Now that all the parameters have been correctly created and configured on the *Analytical Document*, you can execute it.

12. Click on the icon EXECUTE which corresponds to the new *DATA_MINING* analytica document.

13. In the new page you will be required to enter a value for the *DM_CLUSTERER_PAR* and *DM_CLUSTERS_NUM_PAR* parameters.

14.Select *Simple-KMeans* from the combo-box.

15. Insert the number of clusters you want to generate into the input box

16. Click on the EXECUTE icon to execute the document.

17. Data Mining Flow processing can take quite a long time depending on the dimensions of your dataset. For this reason weka engine process all the request in background. This mean that when you execute a data mining analytical document you don't have to wait until the engine finished processing it but you can continue with your work

18. Logout

7.6.QUERY BY EXAMPLE

Query By Example is a SpagoBI tool realized in order to ease the user to create simple queries through a graphical interface.

Notice that this feature is in a RC (release candidate) state.

By clicking on the relative menu item, a list of the QbE queries is displayed.

Figure 72 – QbE: Query list

On the lower side of the window the user can choose to modify an existing query, by clicking on the corresponding EXECUTE icon, or to create a new one by selecting the *here* link at the top of the window.

The composition process is scheduled into eigth steps that will ease the user to create a new query:

- **Select Fields** : to choose the **select** fields;
- **Conditions**: to set the **where** conditions;
- **Ordering**: to select the fields for the **order by**;
- **Grouping**: to select the fields for the **group by**
- **View Query**: to display the query realized following the wizard or to write a proper query using HQL/SQL language;
- **Save Query**: to verify if the query is formally correct and to save it;
- **Execute Preview:** to display the results obtained by the query realized and eventually export them.
- **Generate Template:** to automatically generate a JasperReports report template based on the query.

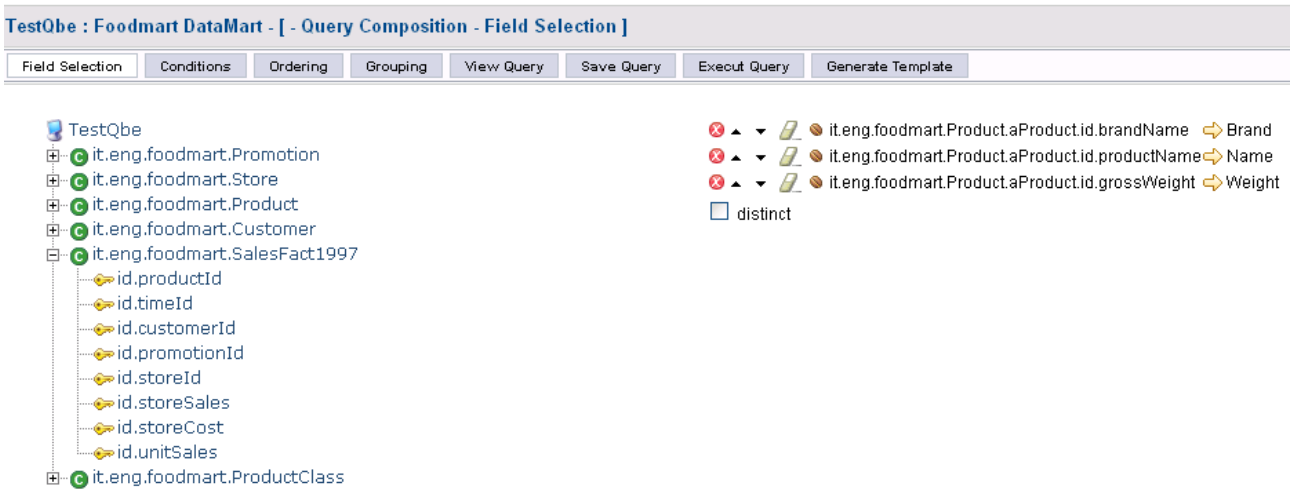

Figure 73 – QbE: Field Selection

7.6.1. Field Selection

On the left hand side of the window a logic view of the datamart is displayed. Therefore, the user can easily navigate the tree and select the desired fields simply by clicking on them.

Figure 74 – QbE: Field Selection detail

Once selected a field, it appears on the right hand side.
The following option are available:

: to erase the corresponding selection;

: to move up or down the field in the list, to set the order of the columns of the result table;

: to apply a group operator on a field, by selecting the proper one from the list;

: to remove a group operator;

: to edit an alias name that will be displayed as header of the column corresponding to the selected field in the result table.

Moreover, the **distinct** option can be set by clicking on the corresponding check below the selected fields.

7.6.2. Condition

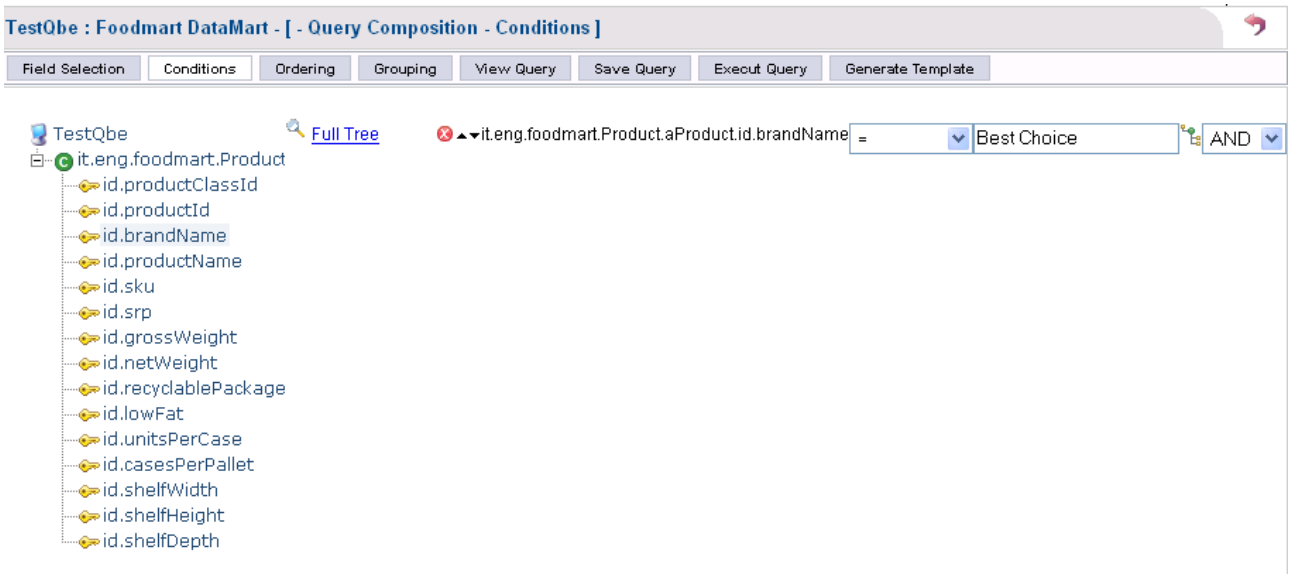

Figure 75 – QbE: Condition

On the left hand side of the window the logic view of the datamart model is displayed.

Notice that to help the user, the Qbe will display only the clauses that contain a **select field**. This is called **Light Tree.** To display the complete datamart tree, the user can simply click on the Full Tree icon.

Once selected a field, on the right hand side of the page, the user can complete the where condition selecting the proper operator, adding the right condition in the text area and choosing the logic operator that will be set before the following condition.

Figure 76 – QbE: Where condition

Notice that the right part of the where condition can also be a field: in this case, the user can simply click on the TREE icon and select the desired field.

Notice that the datamart tree displayed is the same shown to choose the left condition.

Both the **Ordering** and the **Grouping** page display on the left hand side a list of the select fields of the query.
By clicking on the name of one of them, the user can set respectively the **order by** and the **group by** fields.

If the user tries to access one of these pages without having already selected at least a field in **Field Selection** page, the Qbe displays an error message

Each selection can be erased by means of the relative icons:

- To delete
- To sort

7.6.3. VIEW QUERY

In the **View Query** the query realized through the wizard procedure is displayed on the left hand side of the page.

Figure 77 – QbE: View Query

The user can also try to write his own query in the text area on the right hand side of the page. This is called **Expert Composition**.

The **Resume From Query** icon can be used to copy the query automatically realized by the Qbe in the text area in order to modify it.
The user can also **save the expert query** realized and **resume the last expert query** saved simply by clicking on the corresponding icons.

The two radio buttons on the bottom line of the page must be used to set if the default query to be executed is the one realized in the Qbe Automatically Composition or in the Expert Composition.

7.6.4.Save Query

Figure 78 – QbE: Save Query

Once selected the **Save Query** tab, the Qbe will try to execute both the automatically composed and the expert query and will display on the right hand side a message to indicate if each query works properly or not.

On the other side of the page, the user can fill in the form required to save the query.
The user can also specify the scope of the query:

- Public scope: the query will have the same visibility as the main Data Mart object
- Private scope: the query will be visible only to the user

7.6.5. Execute query

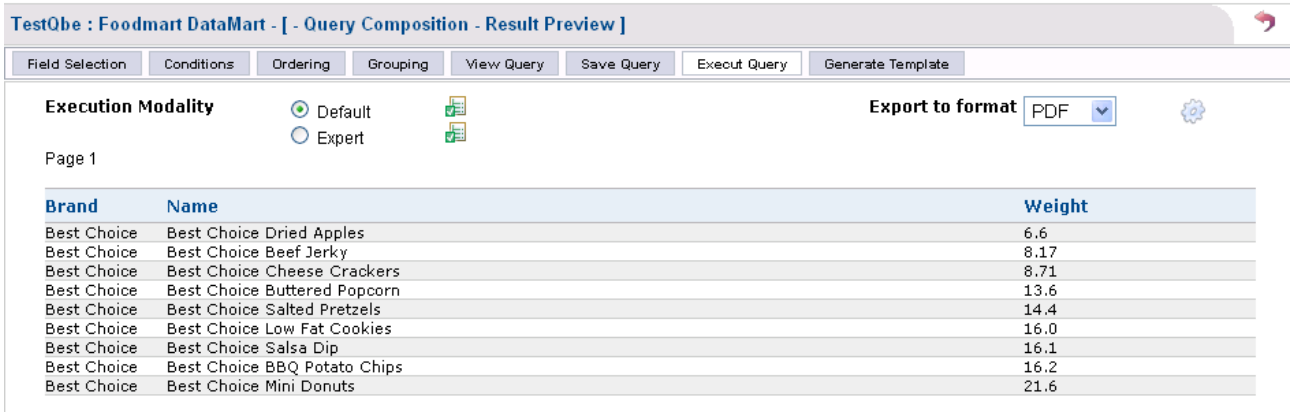

Figure 79 – QbE: Execute query

The **Execute query** page first tries to execute the expert or the automatically composed query, depending on the selection performed in the **View Query** page.

If the query works correctly, a table containing the result set is displayed. Otherwise, a text area containing a description of the error occurred will be displayed.

It is possible to pass from the expert query results to the automatically composed query results or vice-versa using the *execution modality checkbox* situated in the upper left corner of the page.

It is also possible to export results of the query in other formats using the *export options* situated in the upper right corner of the page. The exportation result can be saved to local disk. If the browser have a pre-configurated plugin viewer for the target exportation format the exportation result can be visualized into the browser window before saving.

7.6.6. Generate Template

The **Generate Template** page generates a JasperReports report template (i.e. a jrxml file) based on the SQL query produced from the HQL query. An example of a report built upon this template is shown in the bottom of the page. The generated template can be saved to local

disk, to be used as a base for further modifications, using the save button situated at the top of the page.

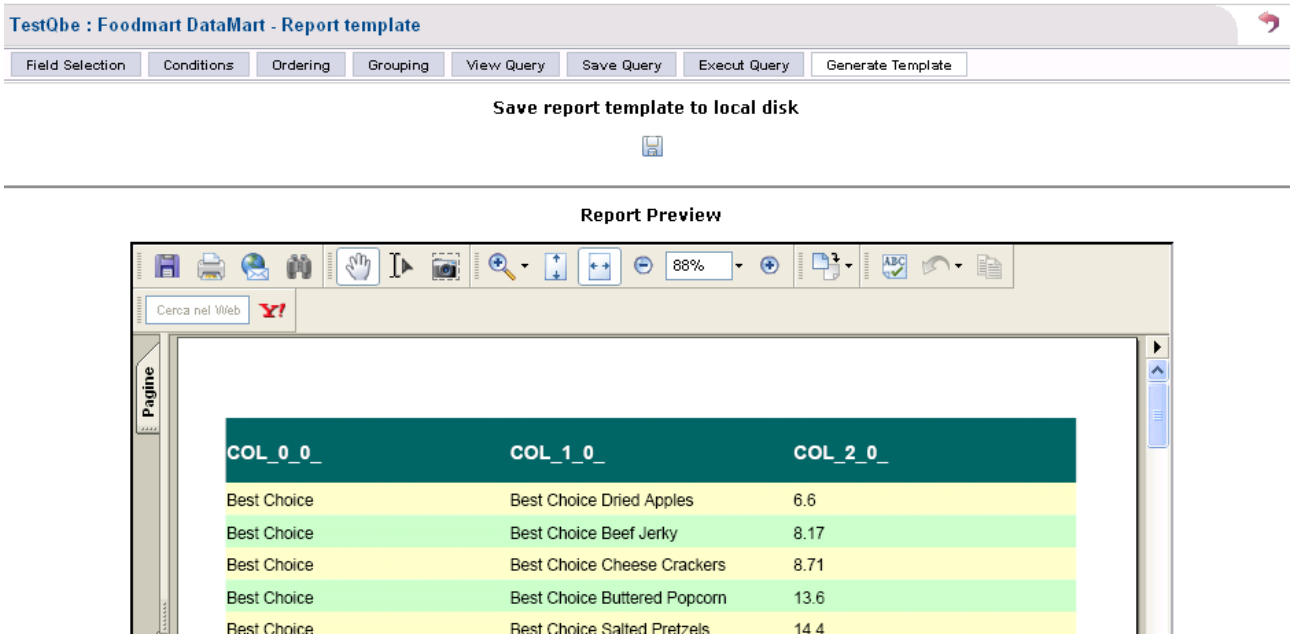

Figure 80 – QbE: Generate Template

7.6.7. Sub-Query

Qbe can handle subquery. The result of a subquery can be used as value in the right term of a where clause. To associate a new subquery to a where clause just click the proper button situated on the right of the choosen clause in the conditions page.

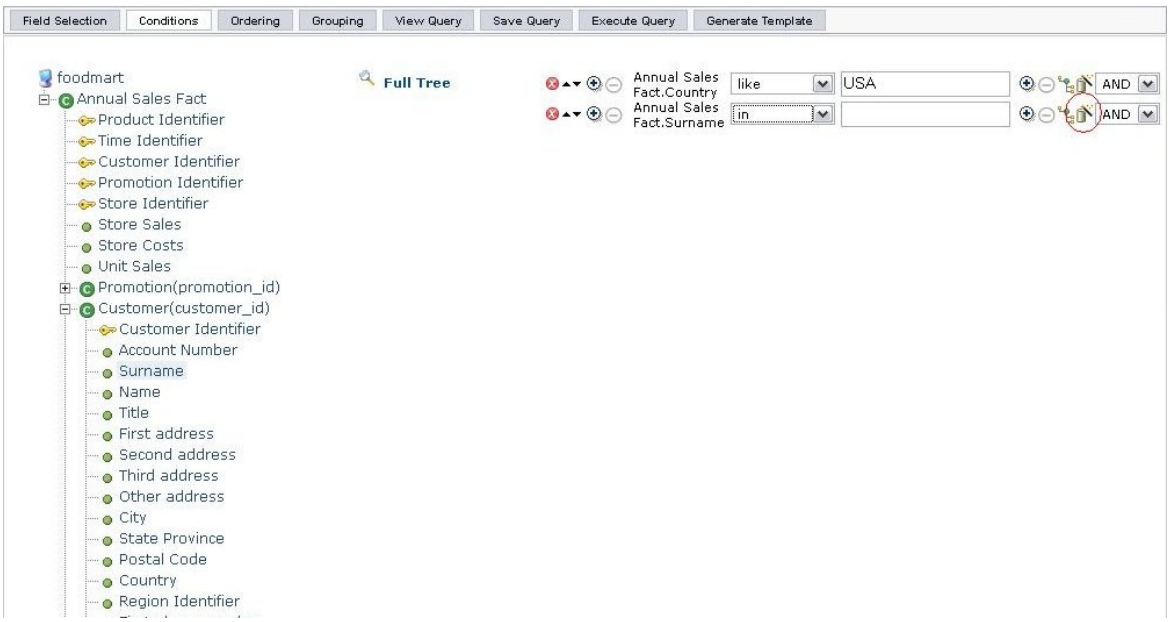

Figura 81 - Sub-query button

A new Qbe window will be popped up. You can now build the subquery in the same way you build a standard query using all the qbe facilites.

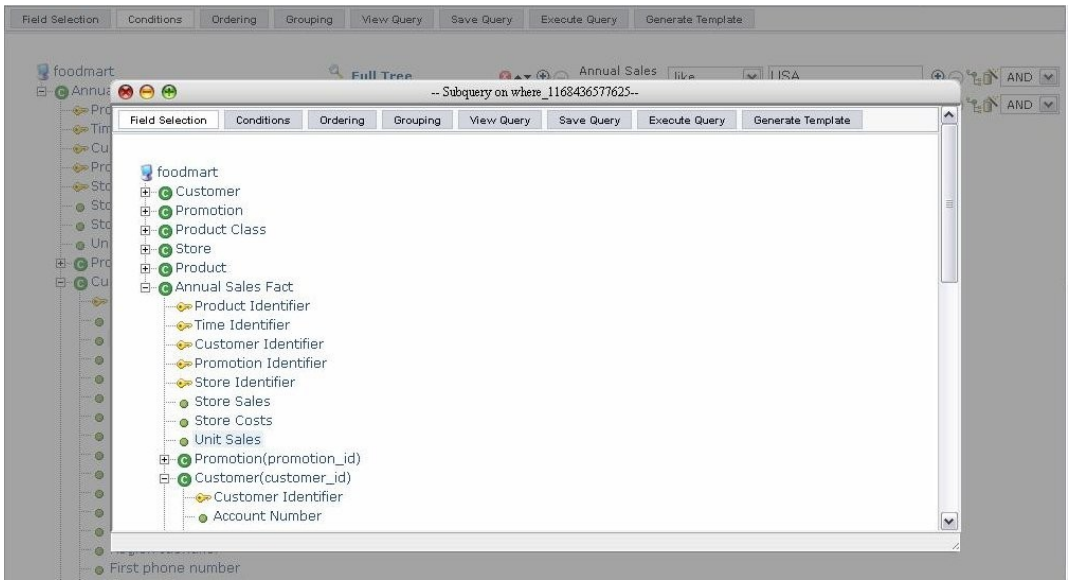

Figura 82 - Sub-query fields selection window

There are anyway two important things to keep in mind when you build a sub query. First, only one field is allowed in the selection clause. This should be the field holding the value or values you want to check against in the selected where clause of the parent query. Second, in the where clauses of a subquery you can reference field appearing in the parent clause through the 'join with parent' button, situated on the right side of each where clause.

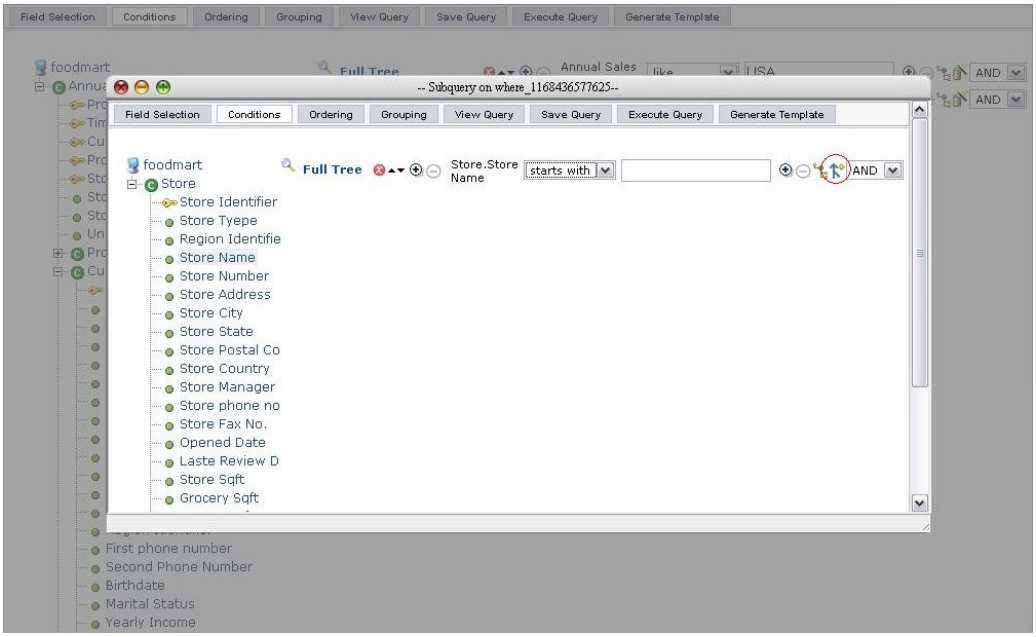

Figura 83 - "Join with parent" button

When you make a reference to fields appearing in the parent query it is not possible to execute, export or save the subquery separated from his parent query. Once you have finished to define the subquery you can close the subquery window to save it.

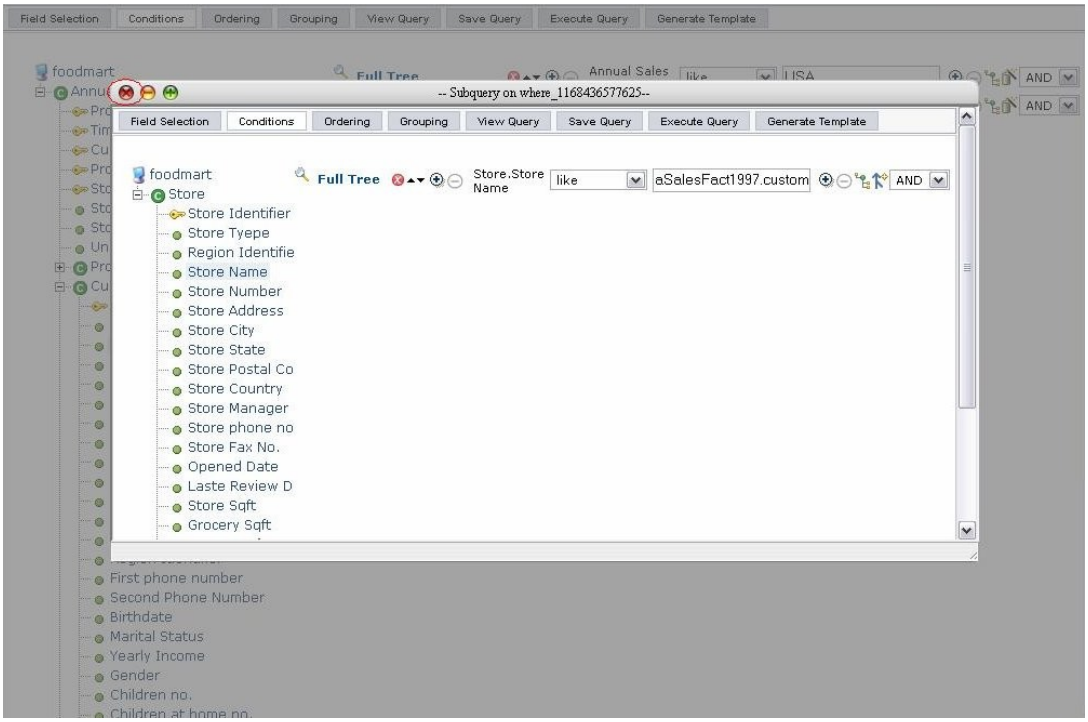

Figura 84 - Sub-query window close button

A saved subquery than can be modified or just displayed at any time using the same button you used to create it.

7.6.8.Create View

Each query built with Qbe can be saved in the database as view using the 'create view' button situated in the up right corner of the 'execute query' page.

Figura 85 - Create a view

The newly saved view are treated by Qbe as a table. Saved views can so be used as a shortcut to define more complex query.

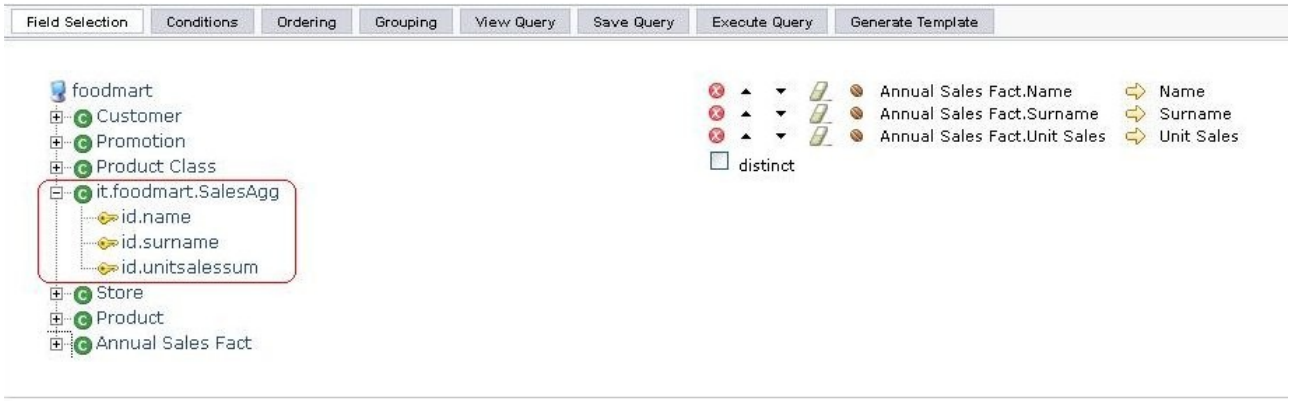

Figura 86 - Created view

7.6.9.Calculated Fields

SpagoBIQbeEngine uses Groovy scripts to generate calculated fields; those scripts must be contained inside the folder specified by the attribute 'dir' of the tag QBE-SCRIPT-DIR in file SpagoBIQbeEngine/WEB-INF/conf.
The scripts must return an object that will be rendered through the toString() method; they can return a String that can be also HTML code, so they can generate a link:
return "...."

Scripts can be modified at runtime, without restarting the server.

To call groovy scripts in your datamart, you have to put an xml file named 'formula.xml' inside your datamart directory. As an example: suppose you have <QBE-MART_DIR dir="tmp/datamarts"/> and <QBE-SCRIPT-DIR dir="tmp/groovy"/>, you must have the following directories structure:

```
SpagoBIQbeEngine/
----------tmp/
----------datamarts/
----------foodmart/
----------datamart.jar
----------formula.xml
----------groovy/
----------customerlnk.groovy
```

where customerlnk.groovy is the file containing the groovy script (different datamarts can use the same script).

This is an example of formula.xml that defines 3 calculated fields:

```
<FORMULAS>
  <FORMULA id="cField1"
     onEntity="it.foodmart.Customer"
     mode="manual"
     mappings="customerId->input1"
     resultingFieldName="Link Detail"
     script="customerlnk.groovy"
     inExport="true"/>
  <FORMULA id="cField2"
     onEntity="it.foodmart.Customer"
     mode="manual"
     mappings="lname->input1,fname->input2"
     resultingFieldName="Link Name"
     script="customername.groovy"
     inExport="false"/>
    <FORMULA id="cField3"
     onEntity="it.foodmart.SalesFact1997"
     mode="auto"
     mappings="customer.customerId->input1"
     resultingFieldName="Link Detail"
     script="customerlnk.groovy"
     inExport="false"/>
</FORMULAS>
```

The 'onEntity' attribute specifies the entity which the calculated field is based on.

The 'resultingFieldName' attribute specifies the name of the calculated field that will be showed in the query execution.

The 'script' attribute specifies the groovy script file to be executed.

The 'mappings' attribute specifies the associations between the properties of the entity to be passed to the groovy script and the script input variables (example based on formula with id="cField2": property 'lname' of entity it.foodmart.Customer will be passed to the script as a

variable named 'input1' while the property 'fname' will be passed as another variable named 'input2'). These script variables can be used in the script with the following sintax: ${variable name} (see below the script example).

The 'mode' attribute can be 'manual' or 'auto'.
'manual' mode means that you will find the calculated field as a new property of the base entity in the entities selection tree.

Figura 87 - Calculated fields, mode manual

When you select the calculated field all the properties that are needed to calculate this field will be automatically selected also (example based on formula with id="cField1": if you select "Link Detail" the property it.foodmart.Customer.customerId will be automatically selected also).

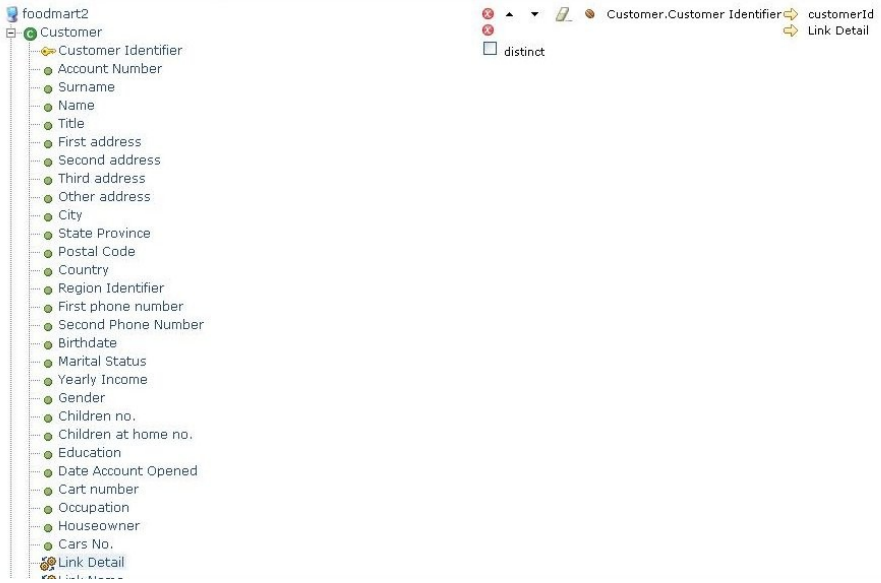

Figura 88 - "Manual" mode calculated field selection

Then, if you execute the query, you will see the calculated field.

Figura 89 - "Manual" mode calculated field execution

'auto' mode means that the calculated field will be always calculated when you select the base entity (the entities selection tree will not show the calculated field if you use this modality); you have also to select on the entities selection tree all the fields which the calculated field depends on,

Figura 90 - "Auto" mode calculated field, all fields selected

Figura 91 - "Auto" mode calculated field

if you don't the calculated field cannot be calculated and a 'N.C.' will be displayed on the relevant column of the query result.

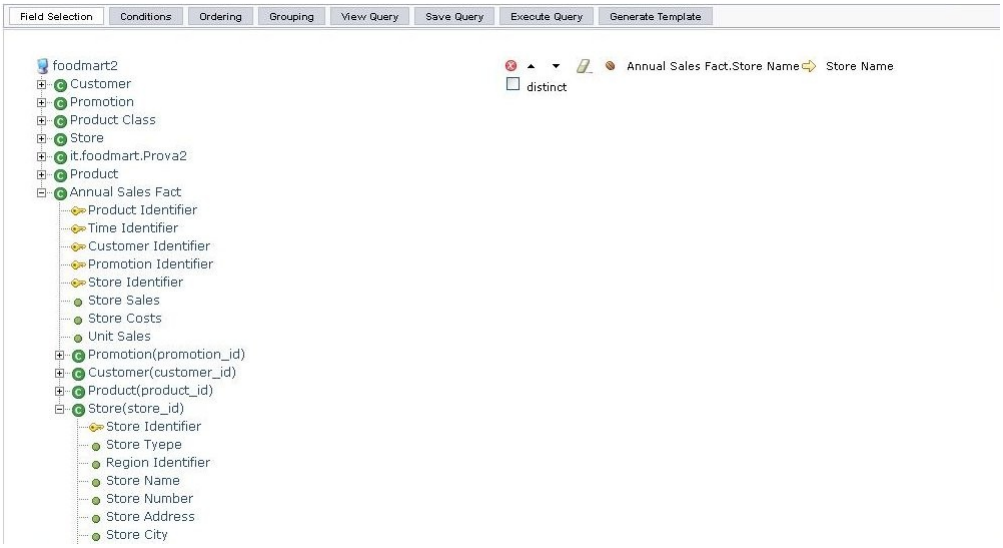

Figura 92 - "Auto" mode calculated field, not all needed fields are selected

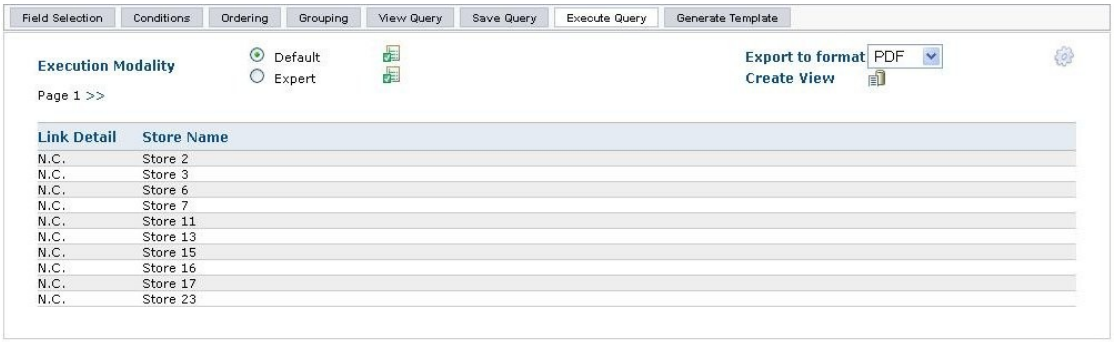

Figura 93 - "Auto" mode calculated field, missing needed fields in the selection clause

The 'inExport' attribute specifies if the calculated fields must be present also in the exported document.

Look at the following groovy script file example:

```
def detailLink(pValue) {
 return "<a href=\"http://…/CUST_DETAIL_ACTION?id=${pValue}\">DETAIL ${pValue}</a>"
}

def detailInExport(pValue) {
   return "PDF [ ${pValue} ] "
}

if ("${qbe_mode}" == "exec"){
    return detailLink("${input1}");
}else{
    return detailInExport("${input1}");
}
```

you can see how to use variables. Yon can see also that there is another implicit variable named 'qbe_mode', that specifies if the query is being normally executing or if it is being exported.

7.7. Maps

A Map business object is a visual representation of some spatial data (typically a geographic map) whose elements is enriched with some graphical information (typically colors) based on the value of some attribute related to the spatial element. Example: a geographic map of the united states with the states filled with different colors based on the population of each one.

If you want to develop a map business object using SpagoBI Geo Engine you need:
- an svg file which contains geographic elements (each one identified by an unique id).
- A database which contains information associated to the ids of the svg map
- A map template file (described below), which contains:
  - the information useful to link the map with the database
  - the styles to apply to svg geographic elements based on the attribute values stored into the database

When the user executes a map document, the system:
- loads the map template file
- reads the database connection configuration
- connects to the database and execs the query in order to extract the data associated to svg geographic elements.
- reads from the template the svg map logical name and gets the map from its catalogue
- scans the map and for each svg element id:
  - get from the result of the query the attribute value related to the id
  - reads from the template file the style assocaited to the value of the attribute
  - Sets the style, extracted from the map template file, to the svg element

Let's make an example:

svg file:

```
<svg>
     <g attrib:name="Europe">
           <path id="1" attrib:name="Italy" d="M0 0 ...." />
      <path id="2" attrib:name="Spain" d="M10 10 ...." />
      <path id="3" attrib:name="Germany" d="M20 20 ...." />
      ...
     </g>
</svg>
```

result returned by the database query:

Ids	*Population*
1	60000000
2	40000000

Ids	*Population*
3	80000000

Template file:

```
<LEVELS>
  <LEVEL threshold="0" style="fill:blue;>
    <TEXT description="less than 50000000" />
  </LEVEL>
  <LEVEL threshold="50000000" style="fill:red;>
    <TEXT description="from 50000000 to 70000000" />
  </LEVEL>
  <LEVEL threshold="70000000" style="fill:green;>
    <TEXT description="from 70000000" />
  </LEVEL>
</LEVELS>
```

The final svg produced:

```
<svg>
     <g attrib:name="Europe">
            <path id="1" attrib:name="Italy" d="M0 0 ...." style="fill:red; />
      <path id="2" attrib:name="Spain" d="M10 10 ...." style="fill:blue;" />
      <path id="3" attrib:name="Germany" d="M20 20 ...." style="fill:green; />
      ...
     </g>
</svg>
```

7.7.1. How to get a geography svg map ?

If you don't have a geographic svg map you can generate it in different ways, for example:

- converting a shapefile into an svg map usign some tools that you can download from internet
- Using some gis/map tools, like for example GeoServer o MapServer.
- Comverting data contained into a postgis database usign the postgis tools for svg convertion
- Extracting data from an opengis database and converting it into svg with a a custom program.

Once you have the svg map be sure that the geometry element ids correspond to the ids contained into your database.

7.7.2. Write a map template file

The template file is divided into some main parts

```
<?xml version="1.0" encoding="UTF-8"?>
<MAP name="name of the map">
      <DATAMART_PROVIDER />
      <CONFIGURATION />
```

```
</MAP>
```

The tag <MAP/> is the root of the template and contains the "name" attribute. The value of this attribute must be the logical name of a map contained into the catalogue of the SpagoBI Geo Engine. To learn how to know which maps are contained into the Engine catalogue or to add a new map you can look at the SpagoBI 'How to' document (question 'How to deploy a new Map into SpagoBIGeoEngine').

The 'Datamart Provider' element is useful to configure the data retrival from your database.

```
<DATAMART_PROVIDER connection_name=" name of a connection defined "
                   query="select id, value from ... where ..."
                   column_id="id"
                   column_value="value"/>
```

You need to configure four attributes:

- connection_name: the name of one of the database connection pools configured into the data_access.xml file (SPAGOBIGEO_HOME/WEB-INF/conf/data_access.xml). The connection pool is used to get the connection towards the database.
- Query: the query which recovers the data associated to the svg element ids
- column_id: the name of the column returned by the query which contains the ids related to the svg element ids.
- column_value: the name of the column returned by the query which contains the value related to each svn element id. The value will be used to access the configuration tag of the map template file and extract the style to apply to the svg element.

The 'Configuration' element contains the styles, that have to be associate to the svg tags based on the database attribute value. The styles are organized following a level hierarchy. Each style is associate to a range of values which is defined by two levels, the first one defines the low boundary while the second one defines the up boundary. When an svg element is analyzed the engine gets its attribute value from the database and using it recover the sytle associated to the low boundary level defined into the map template.

```
<CONFIGURATION>
  <LEGEND x="legend x position" y="legend y position"
          width="legend width" height="legent height" style="legend style">
    <TITLE description="legend title" style="legend style"/>
    <LEVELS>
       <LEVEL threshold="0" style="style of the level">
           <TEXT description="less than 100" style="style of the description" />
       </LEVEL>
       <LEVEL threshold="100" style="style of the level">
           <TEXT description="more than 100" style="style of the description" />
       </LEVEL>
       ...
    </LEVELS>
  </LEGEND>
</CONFIGURATION>
```

Each level tag has two attributes:

- threshold: defines the low boundary for the current level and the up boundary for the previous level.
- Style: the style associate to the level

The level tag contais a 'Text' tag where you can set the description of the level

It's possible to configure a legend of the map (which is based on the level definition). You can configure it filling the attributes of the tags 'legend' an 'title' which allow you to set the x/y position, the dimensions, the style and title of the legend.

7.7.3. Define a Map document into SpagoBI

In order to configure a map document into SpagoBI you need to:

- be sure that you have a engine pointing to the SpagoBI Geo Engine
- define a new business object of type 'Map'.
- Setting into the new document the geo engine
- load the template map file as a template for the map document
- be sure that the Adobe Svg viewer plugin is installed on your browser

If your geo engine is well configured you can exec the document and you get a svg map.

7.8. Office document

An "Office document" is simply a document where the template is a file that can be of various known formats: .doc, .xls, .pdf, .txt, .html … This file can include notes, comments, calculations, images and so on.
An internal engine (Office document Internal Engine) will try to show the file inside the browser page (see the example below with a .pdf file): in case the browser has not the plugin for the correct visualization, the file will be downloaded.
These documents are considered static and parameters are unuseful for them.

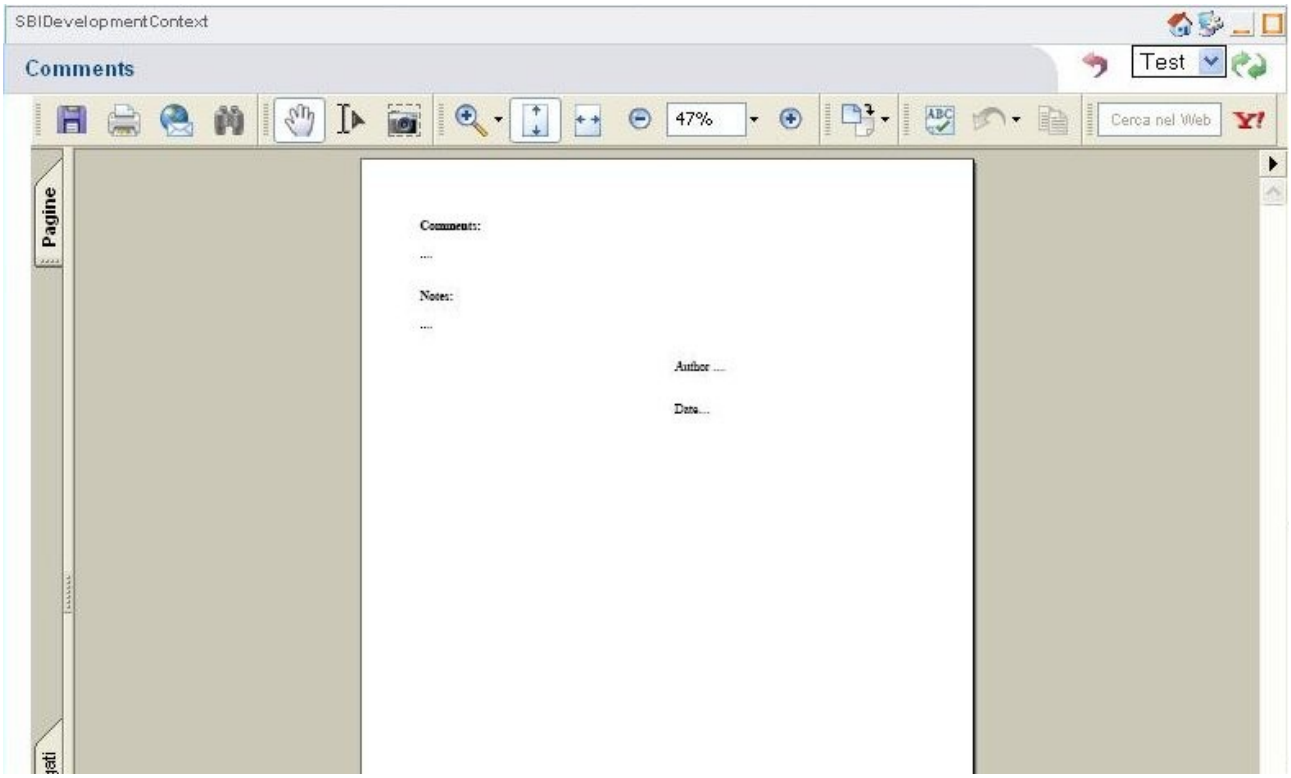

Figura 94 - Office document example

7.9. Booklets

A booklet is collection of report snapshots and relative text notes inserted by SpagoBI users. The final product of the booklet execution is a presentation document (a power point file) which contains, into different slides, the report snapshots and notes. A booklet is a kind of SpagoBI document, but it is a little bit different from the others for two reasons:

- The template of the document is not a loaded file but it is created using a graphical interface accessible from the detail page of the booklet document.
- The execution of the booklet is not a unique request to an engine but it is a process, led by a workflow engine, which involves some spagobi users and automatic mechanisms, in some different activities.

A booklet is a way to share opinion on business documents and produce different presentations of your business areas.

7.9.1. Creation of a new booklet

Since a booklet is a particular kind of SpagoBI document you can create it in the same way as for the other document types (from the list of business documents press the 'insert' button). Once entered the empty detail page of the document choose, from the type select, the 'booklet' option. The system will automatically change the engine to a default internal engine and the file upload form will be replaced by a 'Generate Template' button (if you press the button before save the document it won't work). Then fill the other fields and save the document.

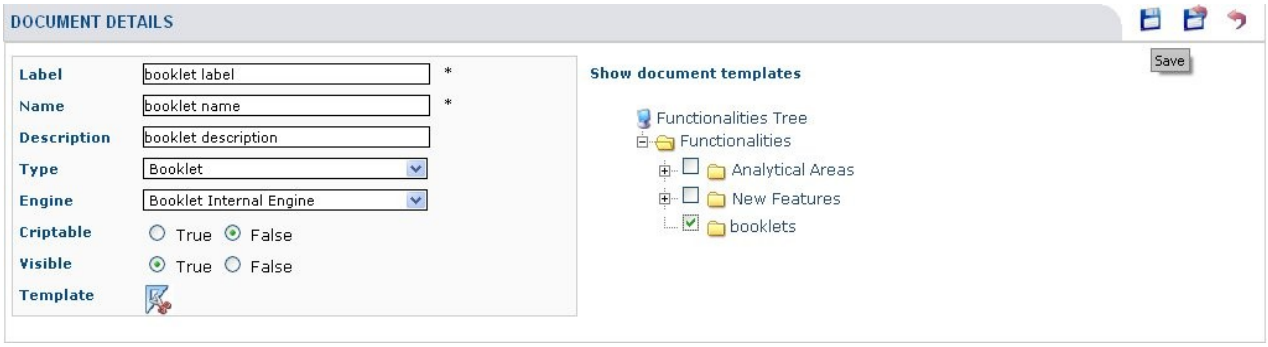

Figura 95 Booklet Document detail page

7.9.2. Configuration of the booklet template

The booklet document doens't allow you to load a file as execution template but give you the possibility to configure a template using a graphical interface (GUI). Using this GUI you can change the current version of the template or you can save a particular configuration as a new version. In the document detail page, as for the other document types, you can see the version template list and set the current one. To access the graphical interface press the button near the 'template' label in the booklet document detail page.

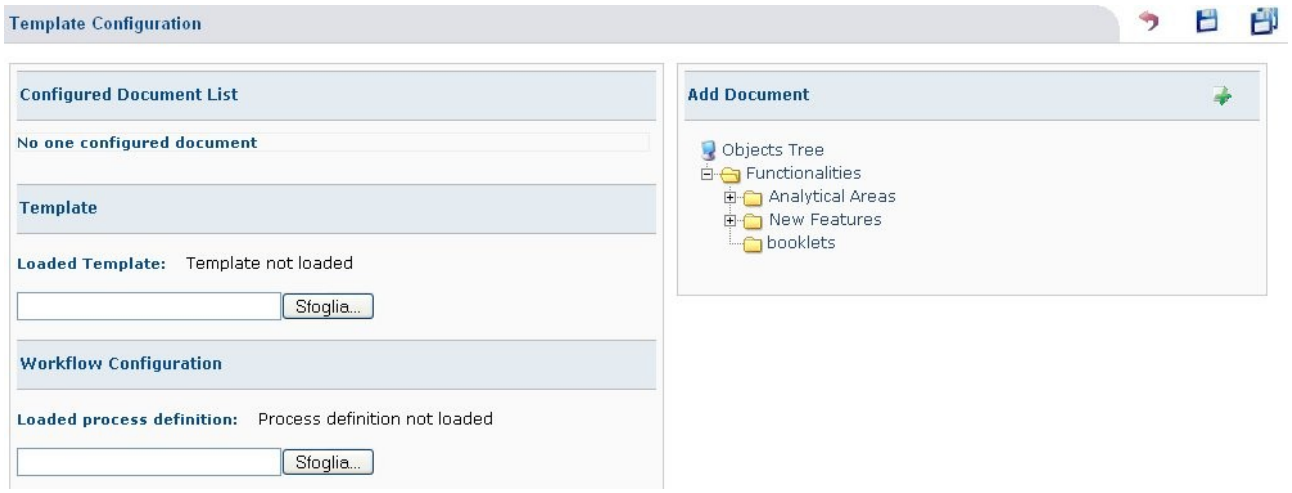

Figura 96 Booklet Template configuration page

The template GUI allows you to:

- Add to the booklet a SpagoBI document. The snapshot of the added document will be inserted in the final ppt presentation. Each document added will be added into the Configured Document List in the left side of the gui. This release allow you to add only jasper reports (document associated to the SpagoBI JasperReportEngine).
- Load the power point template which will be the base for the final document. The template is a power point file (you can create it using the free 'open office' suite) which contains some place holders that will be replaced by the SpagoBI document snapshots. Once loaded the template file it's name will appear near the 'Loaded Template' label together with a button that allow you to download it.
- Add the workflow process definition file. For this version the process definition file must be a jbpm file. Since the booklet module requires a fixed structure for the process definition, in order to change your workflow process, don't create a new definition file but recover the one contained into the BookletModule binary distribution, and then change the allowed parts as described in the section below. Once loaded the workflow process definition it's name will appear near the 'Loaded process definition' label togheter with a button that allow you to download it.

To add a new document to the booklet you have to choose it from the 'Add Document' tree in the right side of the gui (to choose the document simply check it's checkbox), and the press the 'Add Document' button in the right upper corner of the tree box. This operation will get you into the configuration page for the added document.

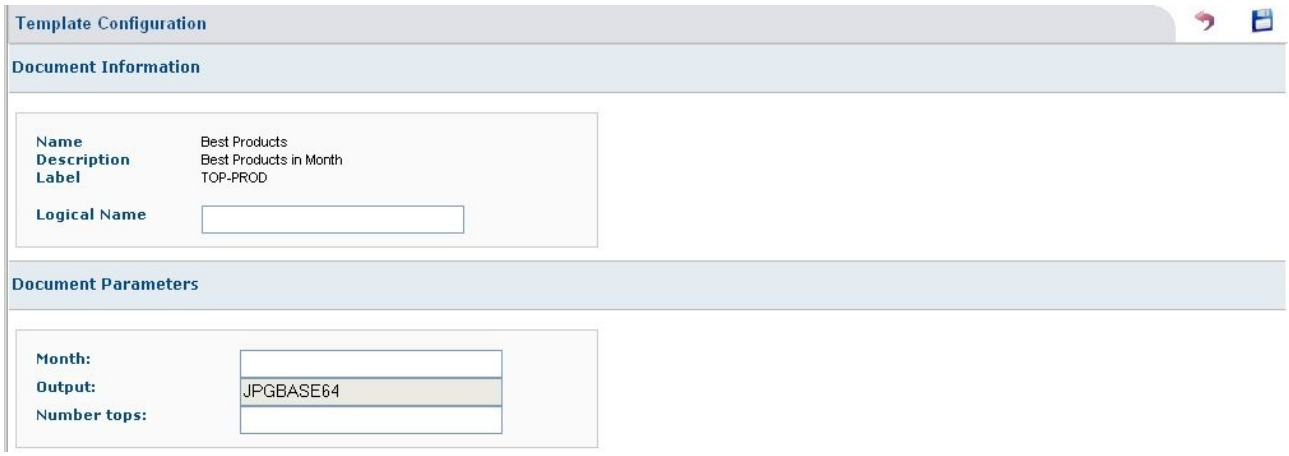

Figura 97 Configuration page for a document added to the booklet

The configuration page contains:

- some summary information about the document chosen (label, name, description)
- An input text which must be filled with a logical name for the document. The logical name is mandatory and must be unique whithin the booklet. It will be used to identify the place holder, into the presentation template, which has to be replaces by the snapshot of the document
- A parameters section where are listed all the parameters of the document together with an input text, useful to specify an execution value. The only parameter value that cannot be changed is the one relative to the output format (it is filled with default value). Remember that the values entered will be used to produce the snapshot of the document which will be inserted into the final presentation.

Once finished the configuration you can save it using the 'save' button in the right upper corner. Then you will return into the booklet template page and the document configured will be shown into the 'Configured document' list (from the list it's possible to erase or modify each document added).

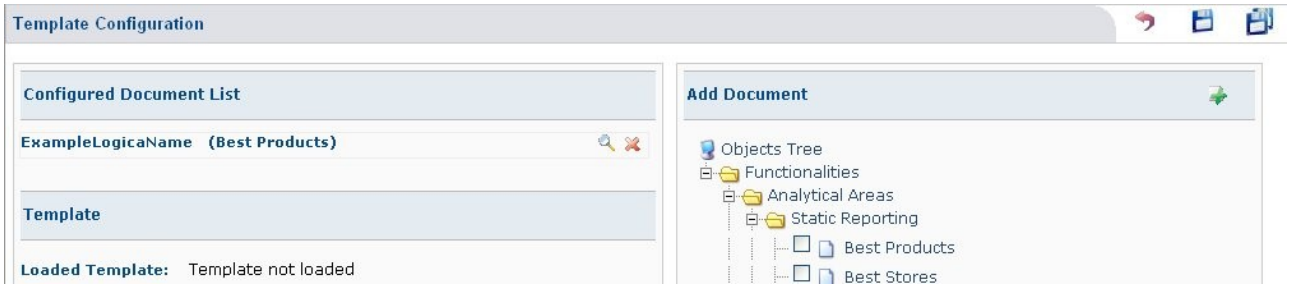

Figura 98 Booklet Configured document list

7.9.3. Create a presentation template

A presentation template is a power point document (remember that you can create it using the free office solution 'open office') which will be the base for the final booklet presentation. You can build your power point template as you like, choosing the number of slides, layout, colors, graphical elements ..., but you have to rember two things:

- When you have decided the position of one document snapshot you have to design in this position a rectangle containing a text string like this: spagobi_placeholder_*logicalNameOfTheDocument.* The final part of the string should be replaced with the logical name of the configured document (look at the previous section). The snapshot of the document will fill exactly the area of the rectangle so rember to choose the right dimensions.
- The comments inserted by the SpagoBI users will be inserted into the notes part of each ppt slide. Configuring the workflow process you have to limit the notes insertion for a particular slide to a SpagoBI group. Only the users belonging to the group assigned to the slide can write notes about the content. This behaviour means that if, in the same slide, you put more than one document snapshots, you have to be sure that each document is visible to the role who view the content of the slide

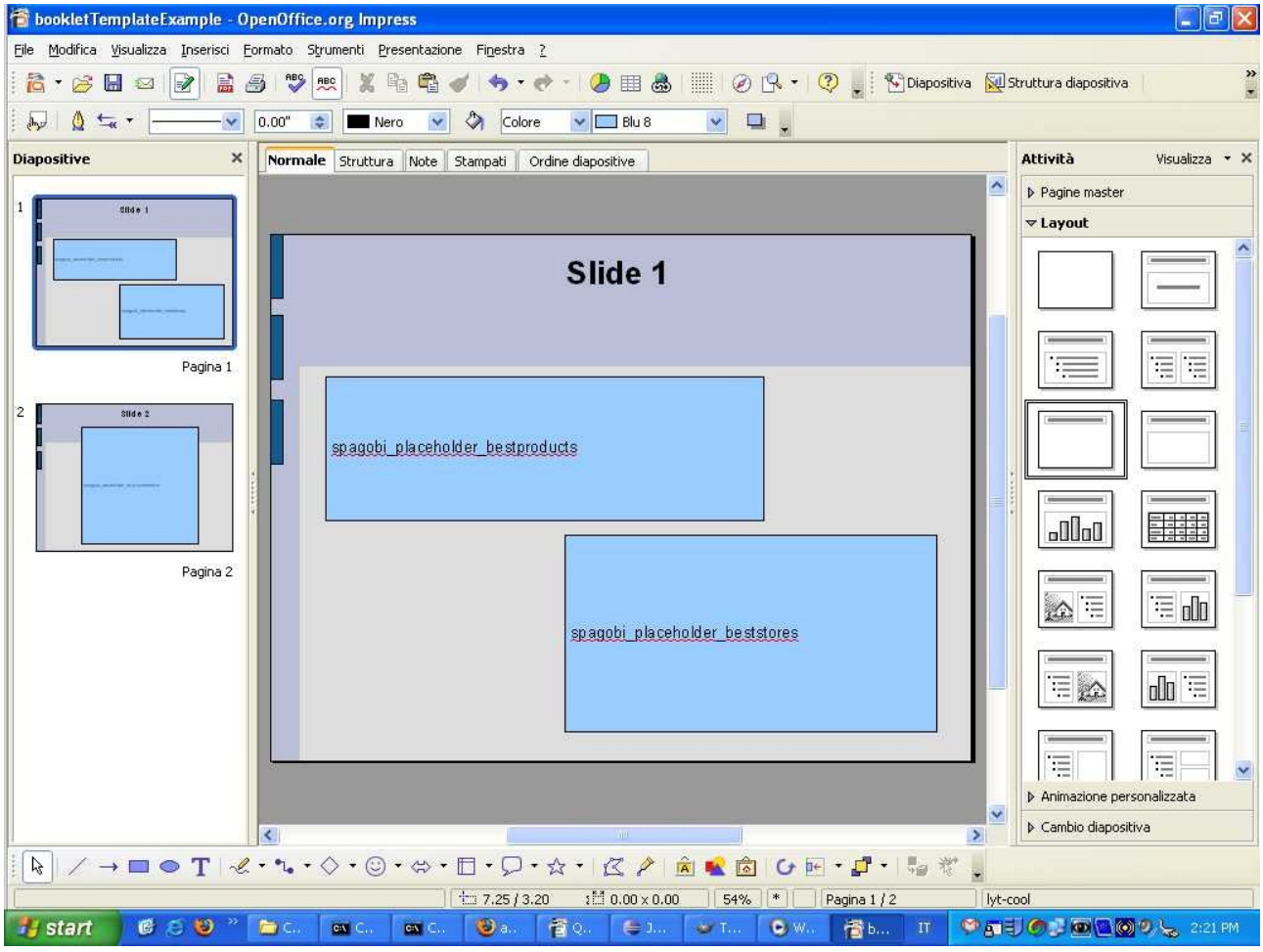

Figura 99 Presentation template

7.9.4. Change booklet woorkflow process

When you exec a booklet document a new workflow process is started. The process has automatic tasks and other performed by the users. The automatic tasks, which analyze the configuration and produce the metadata, cannot be changed. The user task definition can be changed, added or removed, based on some rules listed below. For the current release the workflow process definition must be written using the jbpm sintax. You can find an example

workflow definition inside the booklet module binary distribution. In order to develop your booklet, download the file and change it.

The only points where you can change the workflow definition are:

- the swimlane definition: a swimlane is a role of the workflow. Each task in the process definition is assigned to a swimlane. You can add or remove swimlane definitions but each one of them must have the xml structure reported below (if you add a swimlane place it below the others). You can change only the name attribute giving it as value the name of a SpagoBI role. Swimlane are the bridge between workflow roles and SpagoBI roles.

```
<swimlane name="the name of a spagobi role">
       <assignment class="it.eng.spagobi.booklets.assignmenthandlers.SpagoBISwimlaneAssignment">
       </assignment>
</swimlane>
```

- The 'Add notes' tasks between the fork node and the join node. The xml definition of the task is reported below. The number of this task kind should be equal to the number of presentation template slides containing place holders for spagobi documents. In case you have to add some 'add Notes' task definitions, copy one of the existing one and paste it below the last one. Then you have to add the reference to the task into the foregoing fork node. For each 'add note' you can change the name, the swimlane assignment and the value of the spagobi_booklet_pageindex variable (this variable is the index of the presentation document slide where the notes will be added). Remember that when you change the swimlane name, you are giving the permission to the SpagoBI users who belong to the homonym SpagoBI group, to see the document of the slides and to insert new notes.

```
 <task-node name="give it a name">
      <task name="add_note_task1" swimlane="name of a previous defined swimlane">
         <event type="task-create">
         <script>
               <expression>
                      taskInstance.setVariableLocally("spago_handler", "AddNoteHandler");
                      taskInstance.setVariableLocally("spagobi_booklet_pageindex", " ** ");
               </expression>
       </script>
         </event>
      </task>
      <transition name="" to="join1"></transition>
 </task-node>
```

** index of the ppt document slide where the notes will be pasted

Inside the workflow definition file you find two different activities performed by the user. The first one is the 'Insert notes' activity which is described with the following xml

- The final "Validation Document" task. The xml definition of the task is reported below. This task is the validation of the final presentation document produced. You can only change the assignment to the swimlane.

```
<task-node name="ValidateFinalDocument">
   <task name="validate_document_task" swimlane="name of a previous defined swimlane">
     <event type="task-create">
       <script>
         <expression>
            taskInstance.setVariableLocally("spago_handler",  "ValidateFinalDocumentHandler");
         </expression>
       </script>
```

```
    </event>
  </task>
  <transition name="" to="end1"></transition>
</task-node>
```

7.9.5. Execution of a booklet

Once configured, a booklet is ready to be executed. As for the other SpagoBI document types you can run it from the Objects tree/list. As a first step of the execution the system shows to the user the list of final presentation previously generated. The user can download or delete the previous version from the list or he can start a new presentation version generation.

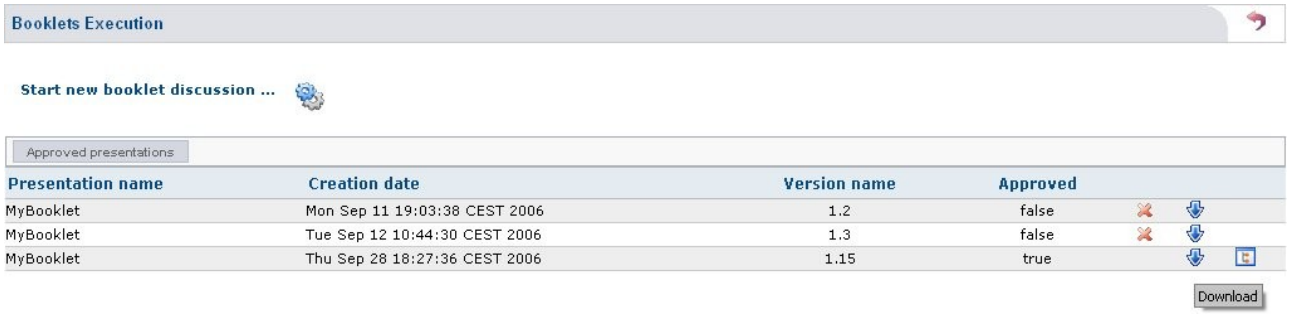

Figura 100 Booklet execution (presentation version list)

Once started a new presentation generation the system initializes the workflow engine, starts the process and prepares the initial metadata. If something goes wrong you will see an error message and you need to contact your system administrator to discover the cause. If the startup phase ends without error you should see a message like this:

Figura 101 Booklet workflow process started correctly

One started, the workflow process executes all the automatic tasks and then assigns to the SpagoBI users the 'Add notes' activities (based on the assigment of the workflow process definition). Each activity is assigned to all the SpagoBI users belonging to the role associated to the task definition (swimlane name). Untill all the assigned 'add note' activities aren't performed the workflow process doesn't continue. In order to know when they have to execute a workflow task the SpagoBI users need to use the 'Worklist' Portlet . This portlet everytime it is refreshed checks if new workflow activities has been assigned to the current user and present them into a list. The user can accept the activity pressing the button on the left of each row. The first user that accept the activity will complete it, all the others will be no more able to accept it.

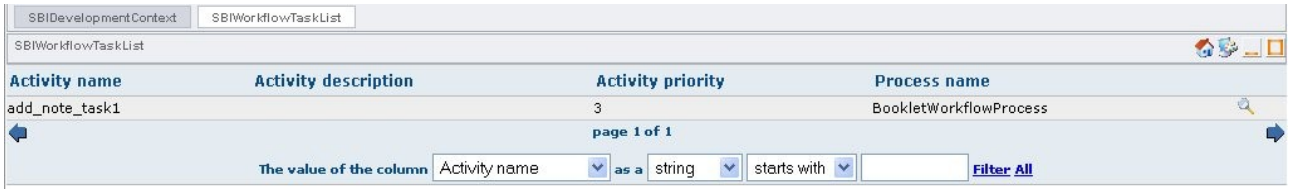

Figura 102 Workflow activities list of the users

Once a user has accepted an 'add note' activity he enters into the add notes editor. The add note editor contains all the snapshot of the document, assigned to the slide associated to the actiity, and a text area for the notes insertion (You can switch from snapshots to text area using the tab in the upper part of the editor). The user can insert different notes and save them. At the end of the note insertion, to complete the activity, the user must press the 'End Discussion' button in the right upper corner.

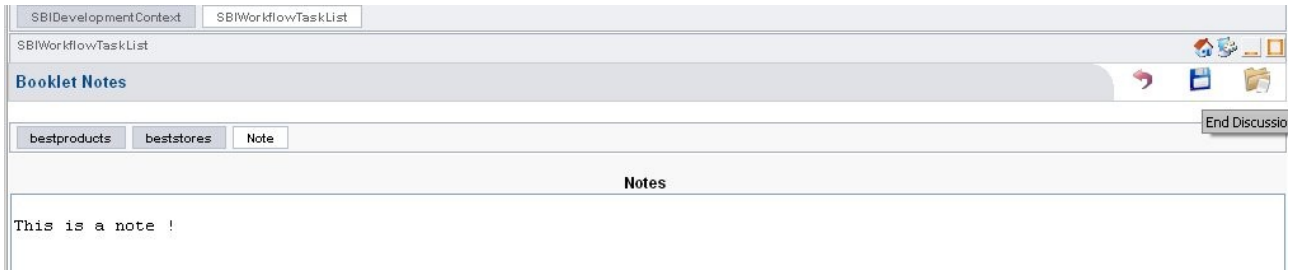

Figura 103 Note Editor

Once all the 'add note' activities have been performed the execution continue producing the final presentation document. The presentation produces must be validated. The validation is the last user activity. The users, who can execute it, will see into the task list a new activity and accepting it they will enter into the validation form.

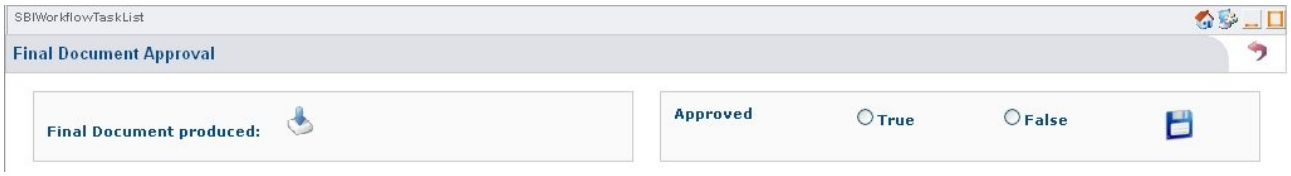

Figura 104 Approve presentation form

Using the validation form the users can download the presentation produced and analyze it. To complete the activity the users must press the save button approving or not the presentation (by choosing the right 'Approved' option). Once saved, the presentation is added to the booklet presentation list (shown when a user exec a booklet), so, if the user execs again the booklet, into the presentation list he will see the new version just created. All the presentations, approved or not, will be saved into the booklet presentation list, but only the approved ones can be deployed, into the SpagoBI BIObject tree, as a document visible to the SpagoBI users. In order to deploy the presentation into the SpagoBI tree the user must press the deploy button, visible only for approved presentations. Once press the deploy button the user enter the deploy form where he can choose the label, name, description and tree brach of the presentation document.

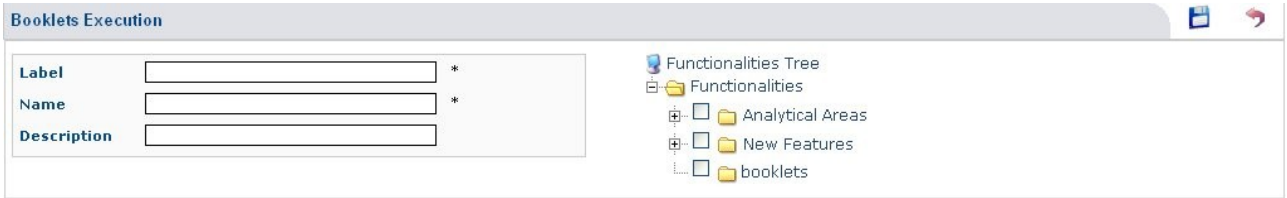

Figura 105 Deploy booklet presentation into SpagoBI objects tree

8. In more depth

8.1. Portal administrator and Portlets organization

Portlets are autonomous and independent application windows. They are freely usable inside portal contexts, supporting the JSR 168 specification, by means of a simple configuration. No development is necessary.

Every function in SpagoBI runs in portlets included into a corporate portal or into a particular Business Intelligence environment.
The portlet organization into the portal is realized by the Portal Administrator.

SpagoBI releases specialized portlets according to the different user typologies (administrator, developer, tester, end-user).
Each user is assigned to a specific typology by the Portal Administrator.

For a better understanding of the user typologies refer to the analytical Document life-cycle section.

8.2. Analytical Document life-cycle

Every SpagoBI document usually follows a three steps life-cycle:

1. **Development**: this is the proper state of every document that has to be developed, corrected, modified or improved, and, therefore, it is the initial state of every new document;
2. **Test**: it is the state of a document which has to be tested in order to check if it works correctly returning the requested result for each possible configuration;
3. **Released**: this is the state of a document that has been properly developed and tested and can be employed by the final user.

Moreover, a 4th state (**Suspended**) can be assigned to a document that will not be used anymore.

Referring to this life-cycle, SpagoBI users can have a specific function which is assigned by the portal administrator.
Users can be classified in 4 different typologies:

1. **Administrator**: he deals with configuration and security aspects.
2. **Developer**: this type of user can create or modify documents;
3. **Tester**: he takes the responsibility to verify the formal correctness of the registered documents and if they fulfil the requirements.
4. **User**: he can use all the business objects in a 'released' state, according to his role and with the modalities previously defined in the parameters configuration.

The *User* is characterized by his functional roles, which regulates:

- the analytical documents visibility;
- the visibility of the data shown by documents;
- the behaviour rules of their parameters and the filters.

It is very important to notice that administrators, developers and testers are also users and, therefore, they can act as specialised users with additional functions.

Every user will access a specialized main page that will contain specific tools.

When completed his own phase, a Developer can update the document state to Test, while a Tester, referring to test results, can change it to Development or to Released.
The administrator is the only one who can modify a document state without any constraints allowing extraordinary maintenance of the documents.
Notice that the simple user cannot modify the document state.

Finally, it is important to observe that in order to develop, test or execute a particular document, it is necessary to have specific rights which can only be assigned by the administrator. For a better understanding of the Security Policy please refer to next paragraph.

8.3. User Roles

Every user is characterized by one or more functional roles.
SpagoBI manages users by their functional roles in order to regulate:

- the analytical documents visibility;
- the visibility of the data shown by documents;
- the behaviour rules of their parameters and the filters.

8.4. Document organization and Security policy

SpagoBI sorts documents in a "Functionalities Tree" which is a File System that can be modified only by an administrator user.

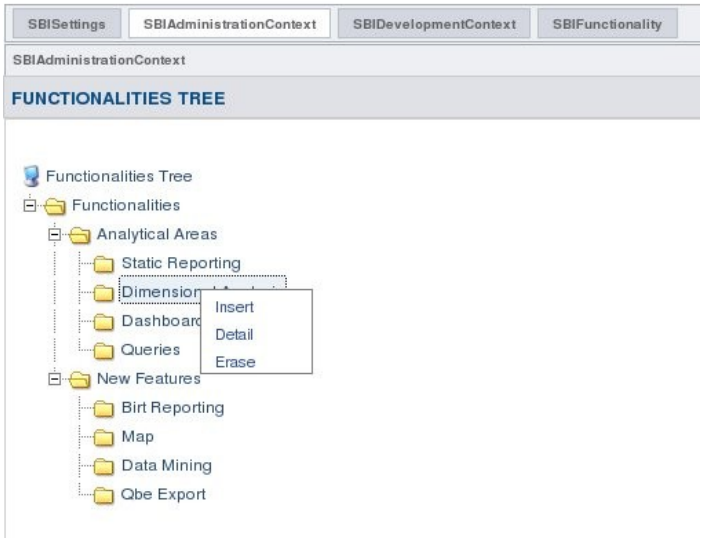

Figure 106 - Functionalities Tree

This allows to better organize documents, grouping them by folders, and to realize a Security Policy. In fact, a user can develop, test or execute a document only if he has at least one role belonging to the corresponding permissions on the folder containing it.
Only an administrator user can set these authorizations for each role and each folder.

For instance, in order to develop a document it is necessary:

1) to be defined as *Developer* by the portal administrator;
2) to have at least a role that belongs the *Development* rights on the folder that contains the document.

To execute a document it is required to:

1) to have at least a role that belongs the *Execute* rights on the folder that contains the document.

8.5. User definition and roles management

Work in progress.

8.6. Portal definition

Work in progress.

8.7.Data Mart (.jar) development for QbE Feature

8.7.1.Brief QBE introduction

The Query By Example (QBE) is a one of the SpagoBI tools which offers a visual mode for data inquiry. It is based on hibernate technology and it presents a database schema as a group of related visual objects. Each database schema is modelled by a set of java classes and hibernate mapping files, all packaged in a jar file.
In order to insert a new database model it's necessary to define a new **datamart** object.
****

In order to recall the datamart object from SpagoBI, you must define a DATAMART type document (using the SpagoBIQbeEngine as external engine) where the template is a simple xml file like the following:

```
<QBE>
   <DATASOURCE name="hsqlFoodmart"
               dialect="org.hibernate.dialect.HSQLDialect" />
   <DATAMART     name="foodmart"/>
</QBE>
```

You have to specify the datasource logical name (as defined inside exo-home/webapps/ SpagoBIQbeEngine/WEB-INF/conf/data-access.xml), the hibernate dialect to be used and the name of the datamart to be executed.

This kind of document does no require any parameters.

In SpagoBI older versions than 1.9.2 the Qbe was an internal engine and the document configuration was quite different.

8.7.2.How to create database model jar file

Since a database model jar is composed by a set of hibernate mapping files and relative java classes it's possible to create it manually, only using java and xml editors. However, this operation can be too long and error prone, so, it's better to use some automatic tools like 'HibernateTools-3.1'.

To get the 'HibernateTools-3.1' working we advise to follow these steps:

- download eclipse 3.1 sdk. It's strongly recommended to install the tools over a new and clean installation of eclipse 3.1. It's possible to download the 3.1 version from the url http://download.eclipse.org/eclipse/downloads/index.php
- install the eclipse platform: simply unzip the zipped downloaded file everywhere you want; a new folder named eclipse will be created (ECLIPSE-HOME)
- download JbossIDE 1.5 final tools file(which contains the hibernate tools). It's possible to download it selecting the JBossIDE-1.5-ALL.zip file from the link http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=72248&release_id=382687
- install the JBoss IDE tools over the eclipse platform: unzip the JBossIDE-1.5-ALL.zip file; a new folder named eclipse will be created (TOOLS-HOME); copy all the contents of TOOLS-HOME/plugins and TOOLS-HOME/features into respectively ECLIPSE-HOME/plugins and ECLIPSE/features folders.

•Start the eclipse platform
•to test if the hibernate tools are working open the view 'Navigator' and then into the navigator box press the right button of the mouse and select 'New/others' menu. Into the new window you should be able to see an hibernate folder(Figure 107 - Eclipse Navigator and Figure 108 - Hibernate options).

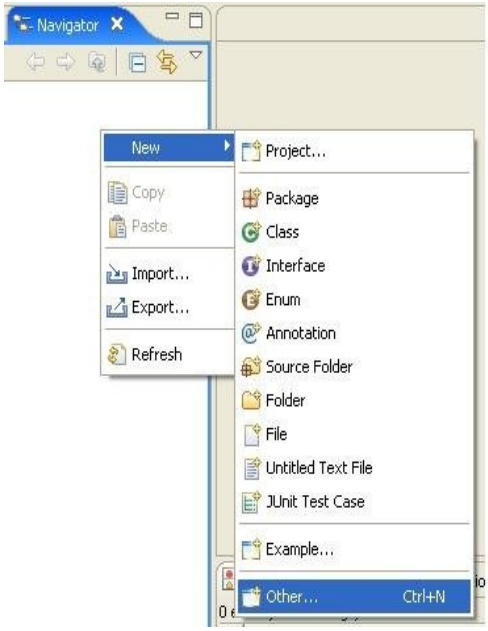

Figure 107 - Eclipse Navigator

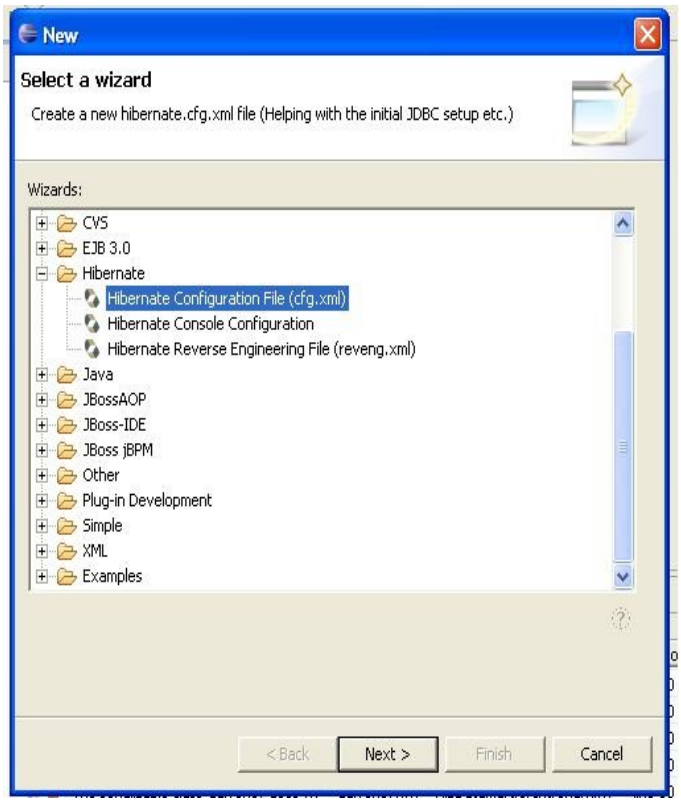

Figure 108 - Hibernate options

Once installed and tested the hibernate tools create a new java project (give it a name and maintain the default properties). Look at Figure 109 - New Java Project and Figure 110 - Default properties of a Java project.

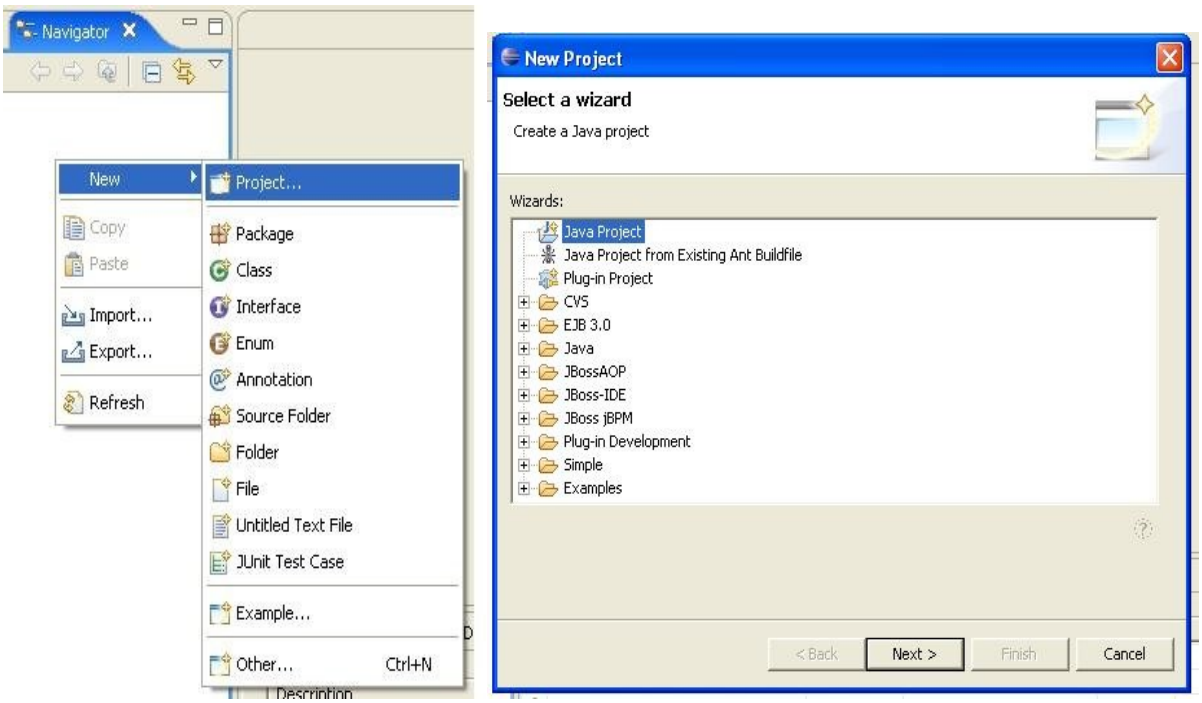

Figure 109 - New Java Project

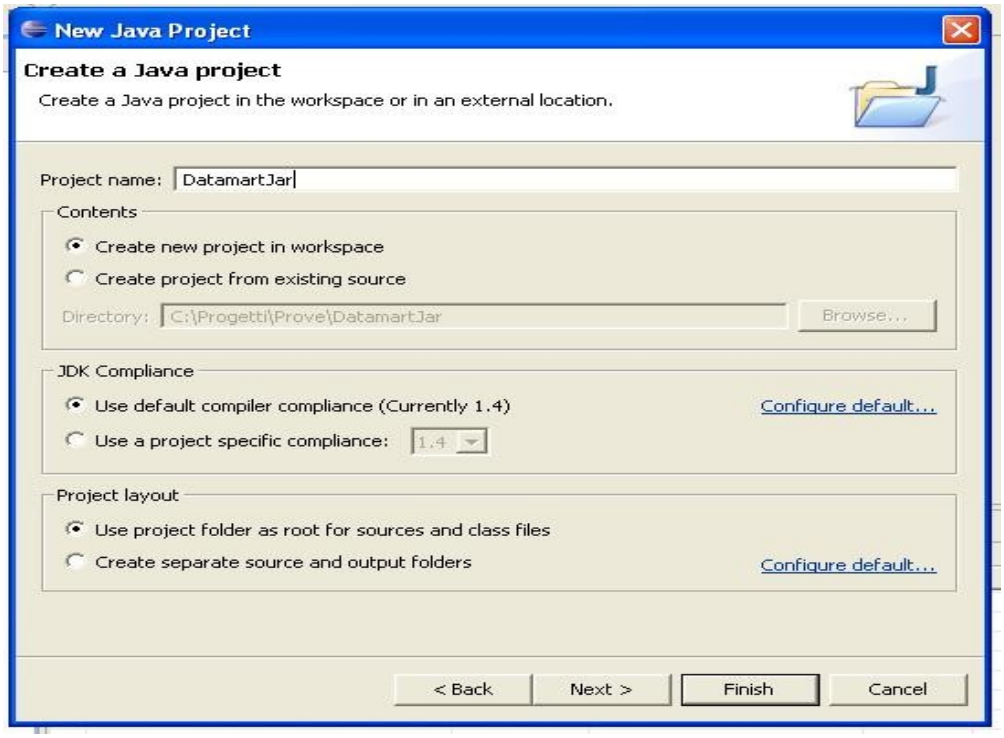

Figure 110 - Default properties of a Java project

Create two project subfolders called 'src' and 'bin'; enter the project properties and select the 'Java Build Path' menu. Into the properties window set the src folder as source folder and the bin folder as output folder (Figure 111 - Java project build path options).

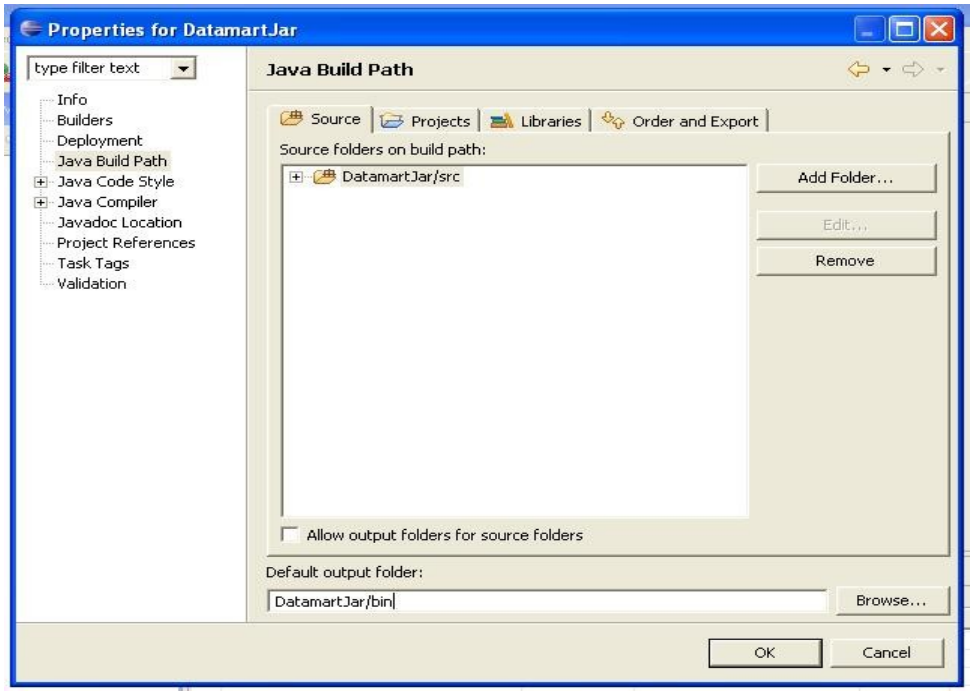

Figure 111 - Java project build path options

create a new Hibernate cfg file into the src folder of the project (right click on the project and then select the 'new/other/hibernate/hibernate cfg' file menu item). Into the properties window choose the right dialect for your database, insert the driver class name, the connection string, user and password. Remember to check the 'create console configuration' option and then press the next button (Figure 112 - New Hibernate Configuration File and Figure 113 – New Hibernate Configuration File).

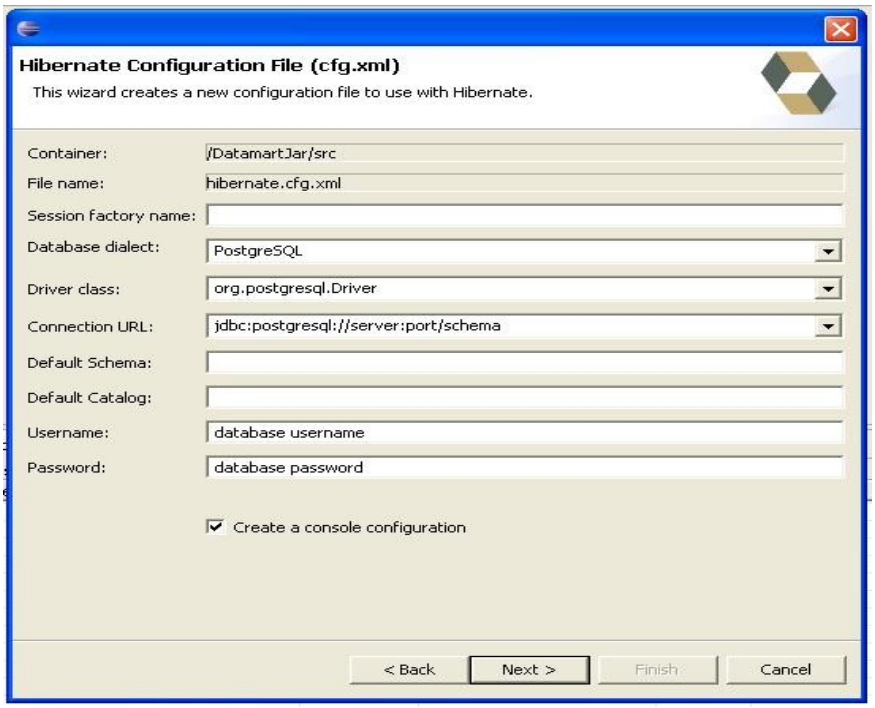

Figure 112 - New Hibernate Configuration File

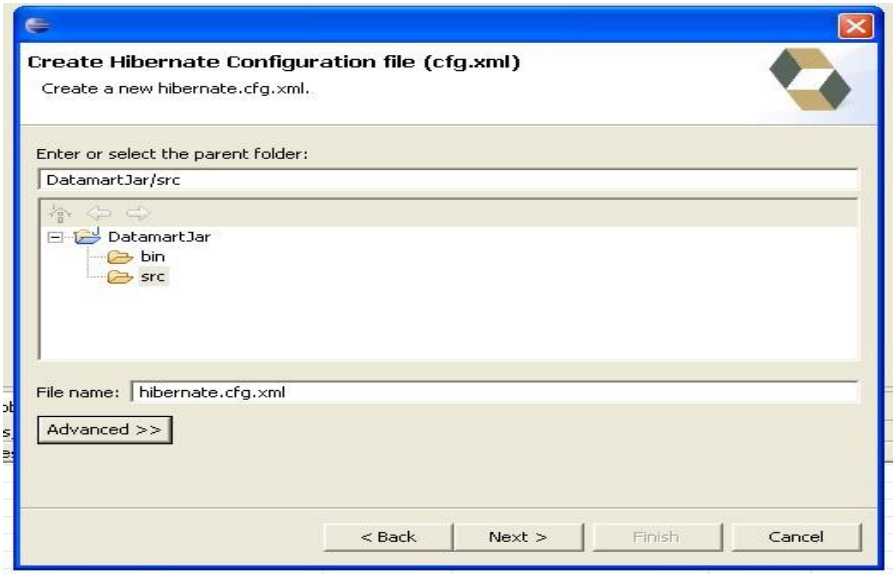

Figure 113 – New Hibernate Configuration File

Give a name to the console, add the jar of the database driver into the classpath and then press the finish button (Figure 114 - New Hibernate Console Configuration)

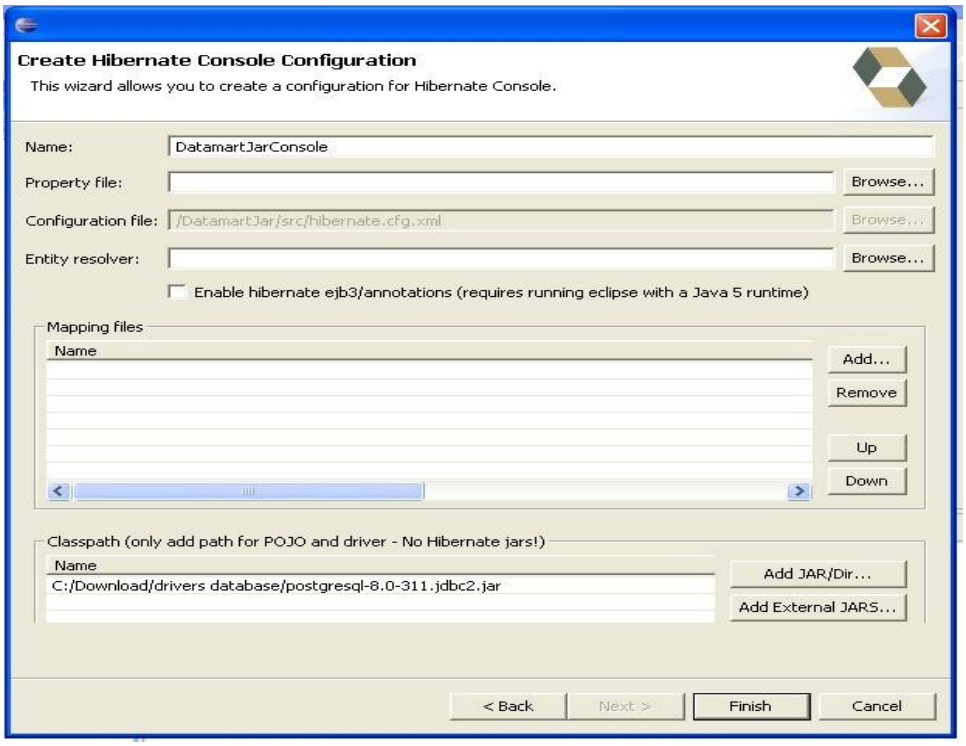

Figure 114 - New Hibernate Console Configuration

Exec the hibernate code generation tool (Figure 115 - Hibernate code generation)

Figure 115 - Hibernate code generation

In the main tab give a name to the generation, select as output folder the src folder of the project, check the 'reverse engineer from jdbc connection' option, give a package name for the generated java classes and fill the 'reveng strategy' field with a '**org.hibernate.cfg.reveng.DelegatingReverseEngineeringStrategy**' value (Figure 116 - Hibernate code generation main options).

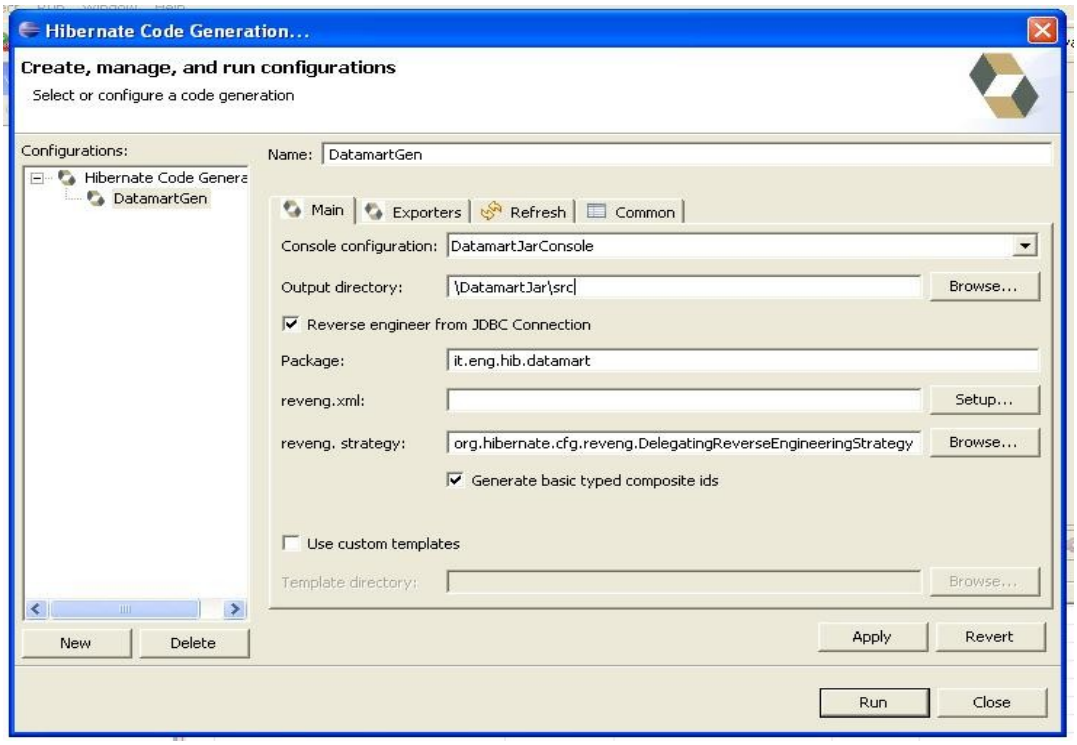

Figure 116 - Hibernate code generation main options

If you want to filter the tables to process for the generation click on the 'setup' button and create a new reverse engineering file. Into the new window select the 'refresh' button and wait until the schema of the database is showed into the left box. From the database schema select the table you want to include and then press the include button. All the tables in the right box will be processed by the tool.
Press the 'Exporters' Tab and check the two options as in the Figure 117 - Hibernate code generation exporters options. The "Generate hibernate configuration (hibernate.cfg.xml)" must be un-checked.

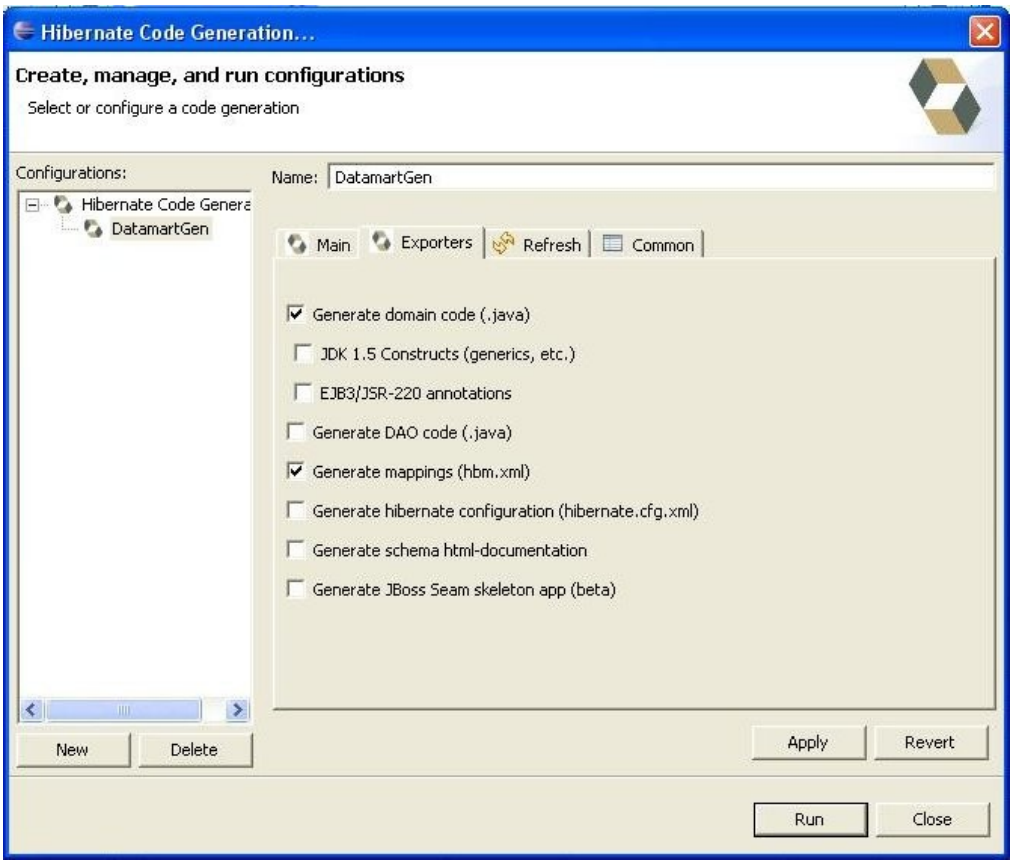

Figure 117 - Hibernate code generation exporters options

At the end Press the 'Apply' button and then the 'Run' Button. After the end of the execution into the src folder of the project there will be a set of java and hibernate mapping files (contained into the right package folder structure). See Figure 118 - Java classes and Hibernate mapping files.

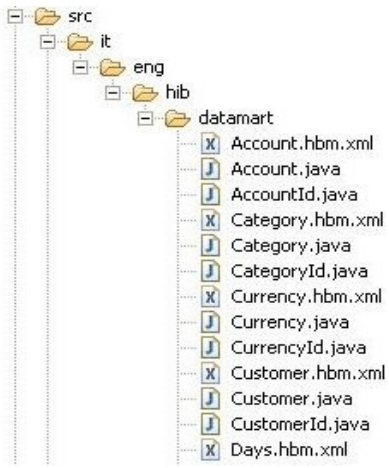

Figure 118 - Java classes and Hibernate mapping files

If the database is well-defined with all the appropriate primary and foreign keys there's no need to change the code produced, but, if some problem occurs, or you are an hibernate expert and you want to personalize the code, it obviously possible to edit each file and make the changes.

By default the qbe tools will show the objects into the graphical interface using the native names of the database table, adding to it the complete package name of the hibernate classes. This behaviour can be changed adding to the source directory a 'label.properties' file. This file must contain a set of association between the native object name and the label to show into the graphical interface. An example of this file is listed below:

```
class.it.foodmart.Promotion=Promotion
class.it.foodmart.Store=Store
class.it.foodmart.Product=Product
field.customerId=Customer Identifier
field.accountNum=Account Number
field.lname=Surname
```

The part on the left side of the equal sign is the complete name of the hibernate object (with a prefix class for the tables and field for table fields). The QBE tool will use the right part of each row as a graphical label for the correspondent object. The file 'label.properties' is the default one and if you need to define your labels in only one language it's enough, but, if you want to translate each label in different languages, you have to define one properties file for each language. Each file must be named with the following rule:
label_<<ISO country code>>.properties
So the files for english / italian labels will be named:
label_en.properties
label_it.properties
Qbe will automatically choose the correct file based on the portal or browser language. (in case there isn't a suitable file it will use the default one).

To create the jar file first compile the project selecting the Projects/clean menu (Figure 119 - Compile project)

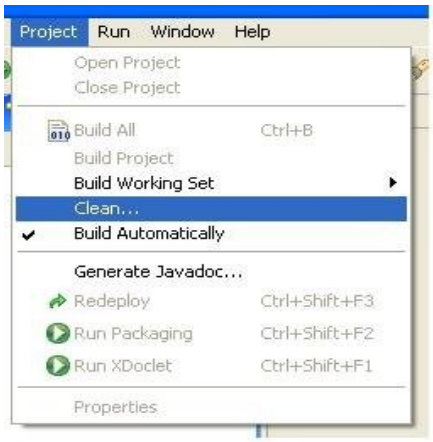

Figure 119 - Compile project

and then export the project as a jar file. Into the export window properties select the src folder of the project, give a destination for the produced jar and then press the 'next' button, 'next' button another time and 'finish' button at the end (Figure 120 - Export jar file).

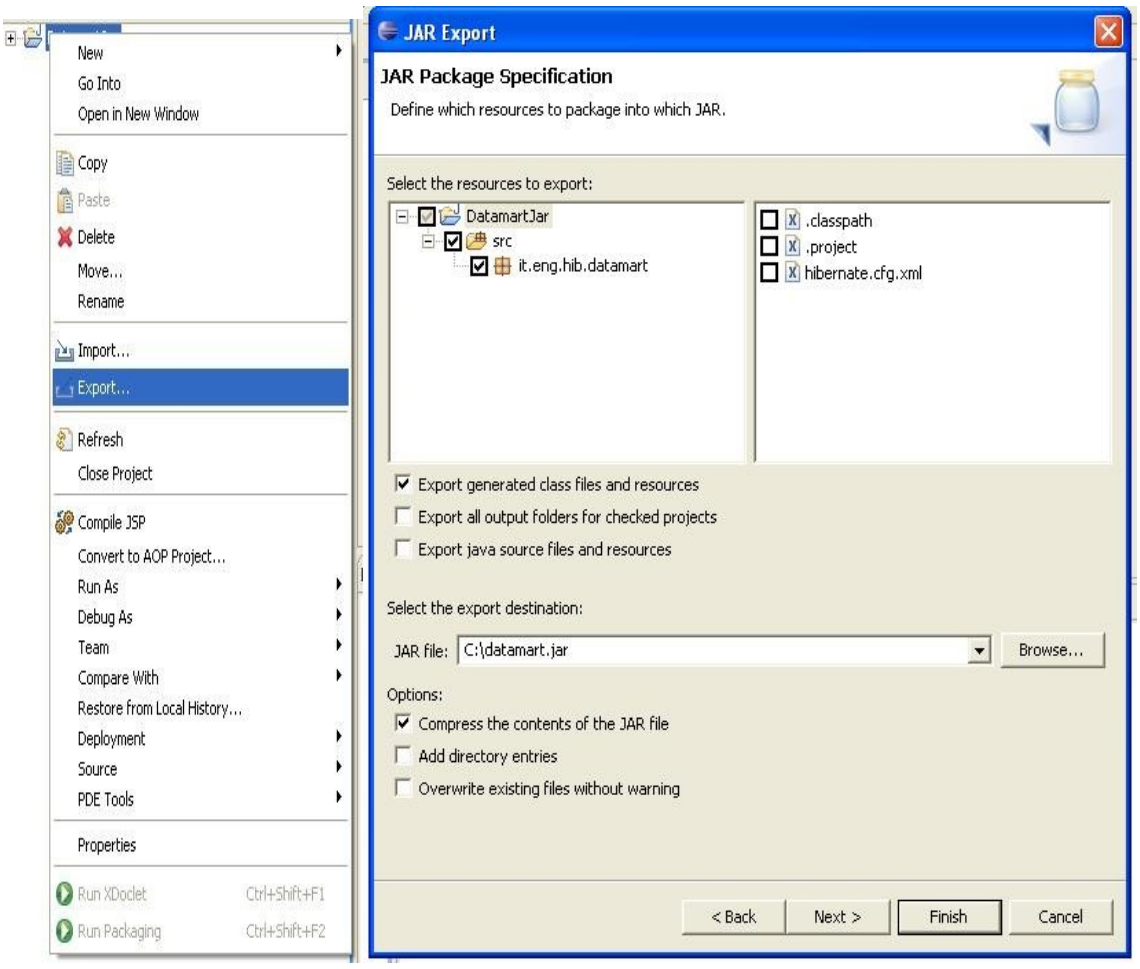

Figure 120 - Export jar file

The datamart jar model can be taken from the destination directory and loaded into a SpagoBIQbeEngine as a new datamart object.

9. Glossary

Work in progress.

Analytical document
Portal
Analytical portal
Parameter
LOV (list of values)
User role
Report
OLAP
Data Mining
Dashboard
Scorecard