# SpagoBI QuickStart

Authors Grazia Cazzin
Luca Scarel

# 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:** | May, 17th 2006 |
| **Update description:** | Third release – Draft (Added paragraph on datamart jar creation) | | |

# 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.
- **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 report template using the iReport tools, you are introduced to the document parametrization and configuration following a logical path. Issues regarding Portal e System configuration are intentionally placed at the end of the chapter, as they involve more complex topics. The chapter uses a static simple report example to explain the SpagoBI main concepts.
- **In more depth**. How to build an analytical portal in more detail.

## 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*

# Help for lecture

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

## 1.1 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. |

## 1.2 Special sections


Note


Example


Reference to other section


In revision phase


Future implementation. To be done.


Advice for the reading of the section

# 2 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.

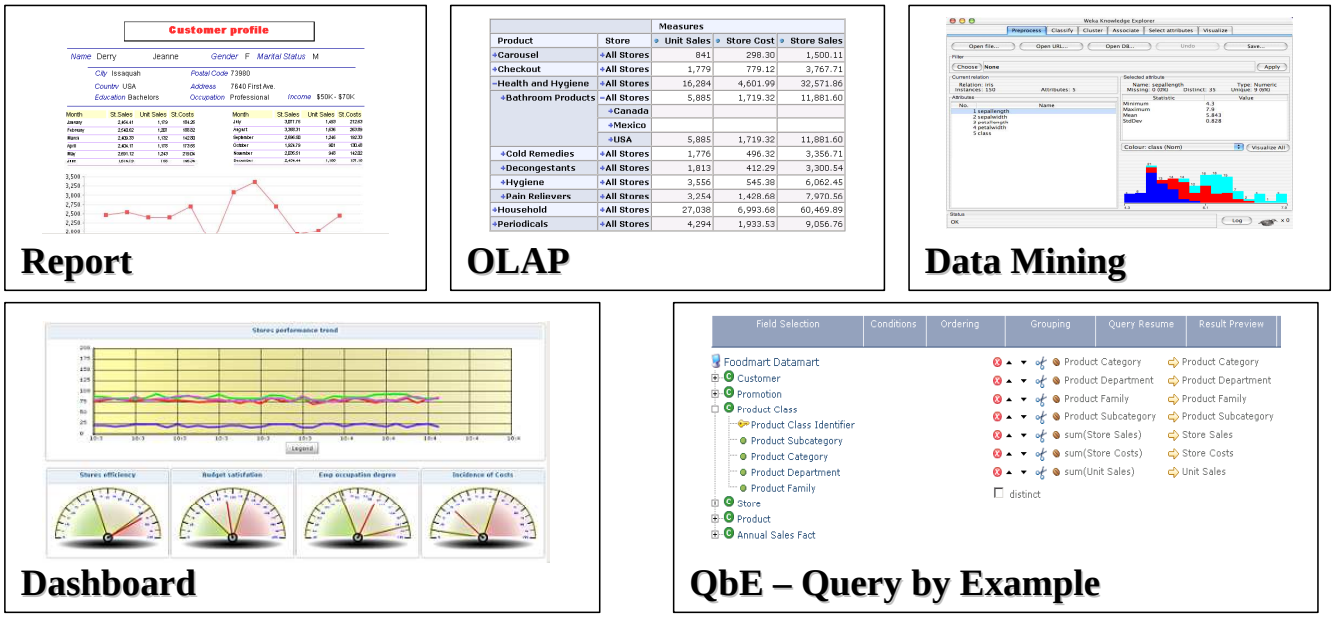



**Fig. 1 – Analytical documents**

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.

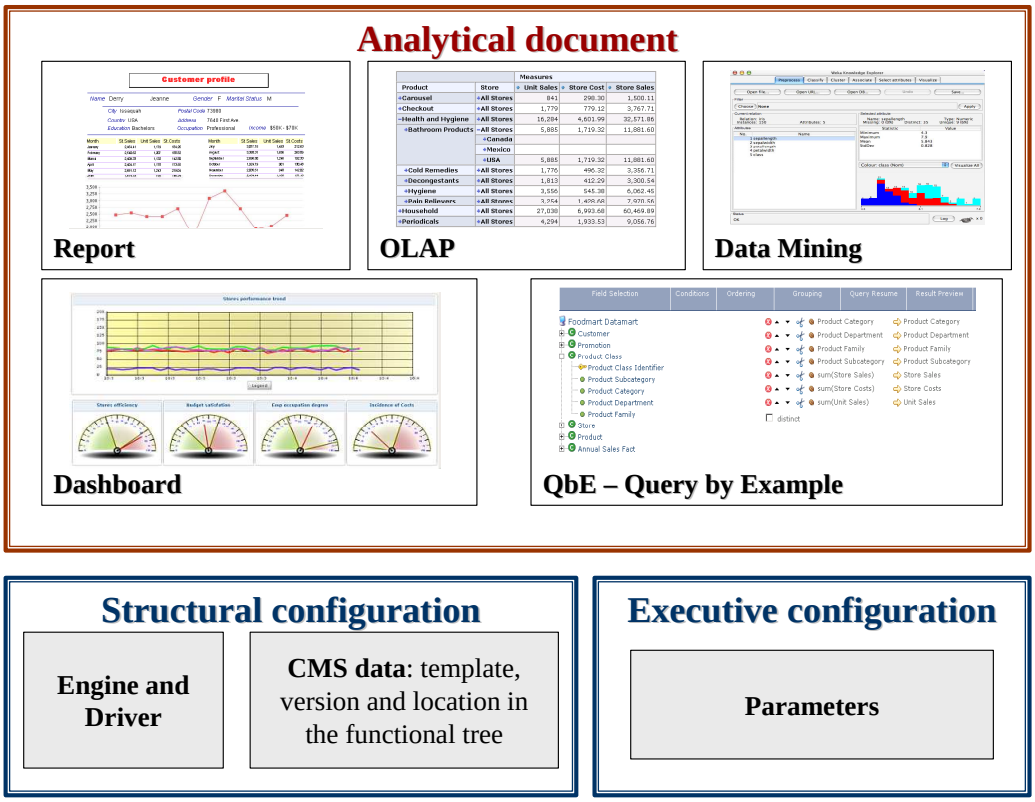



**Fig. 2 – Analytical documents support**

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;
- OLAP on the Mondrian engine, with Jpivot interface;
- Dashboard on the OpenLazslo engine;
- free inquiry on a Hibernate implementation.

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

Data Mining integration is a planned activity.

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.

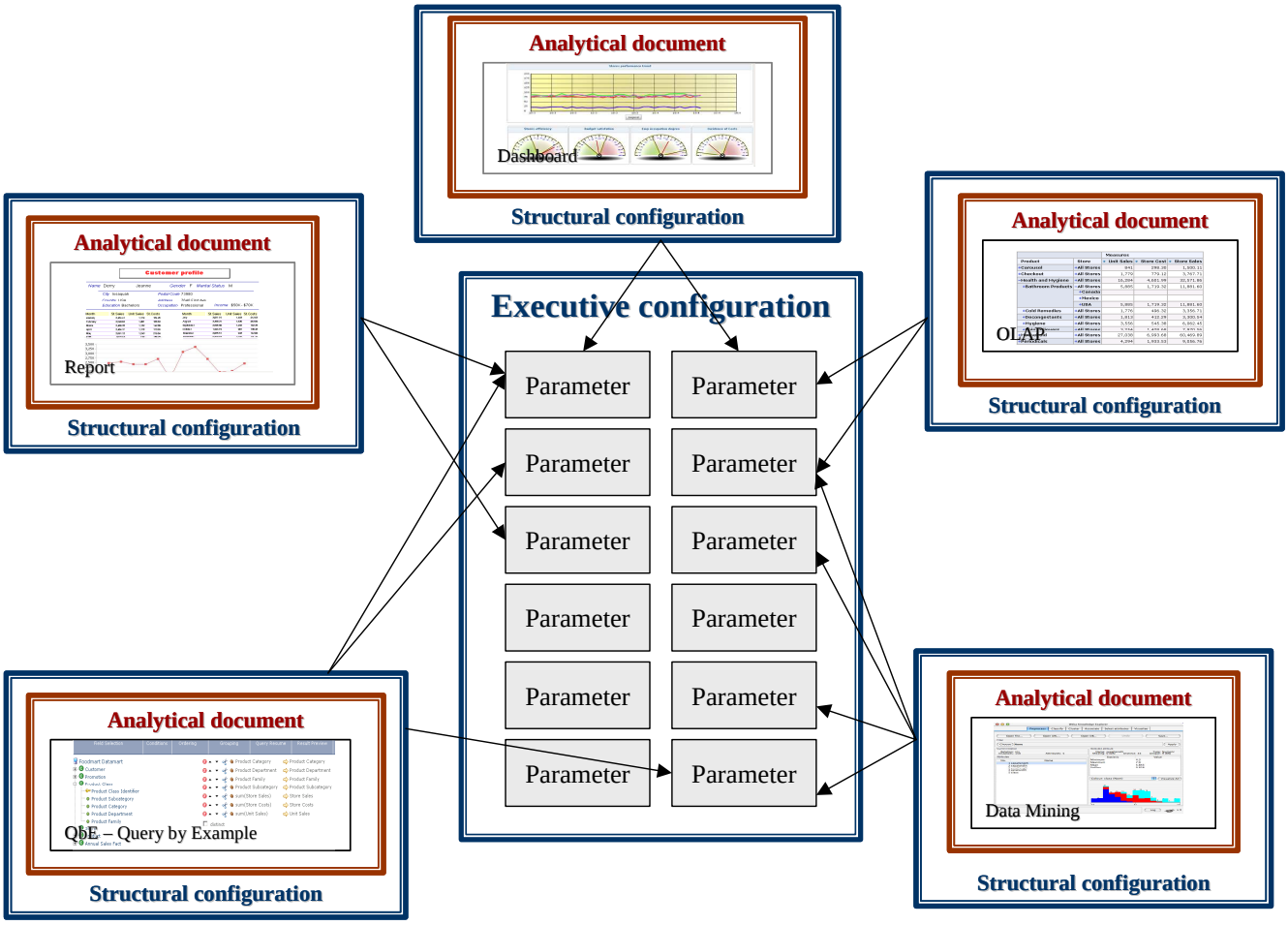



**Fig. 3 – Executive configuration**

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.

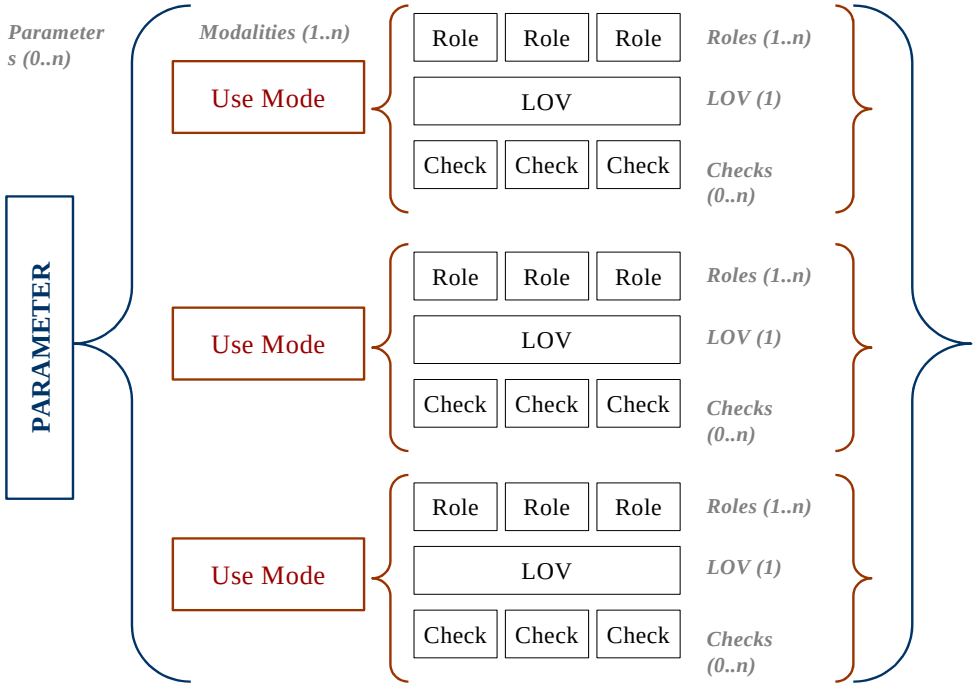



**Fig. 4 – Parameter's abstract structure**

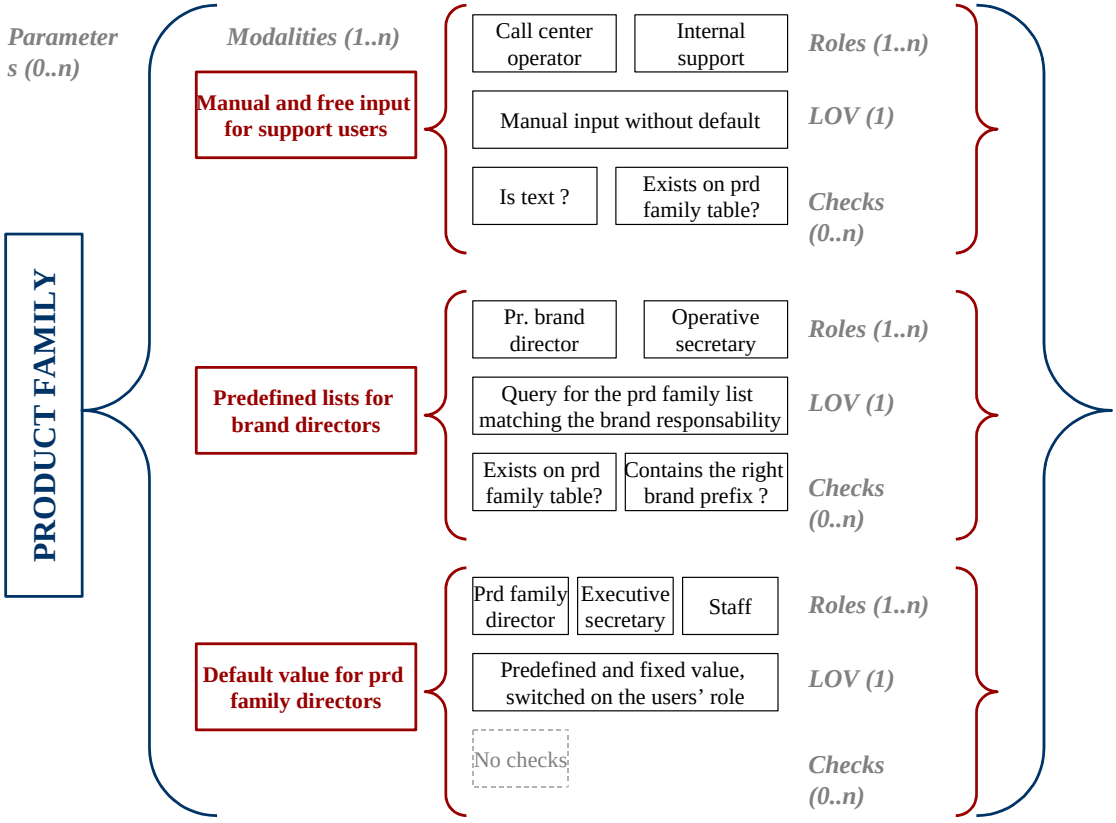



**Fig. 5 – Parameter's implementation example**

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:

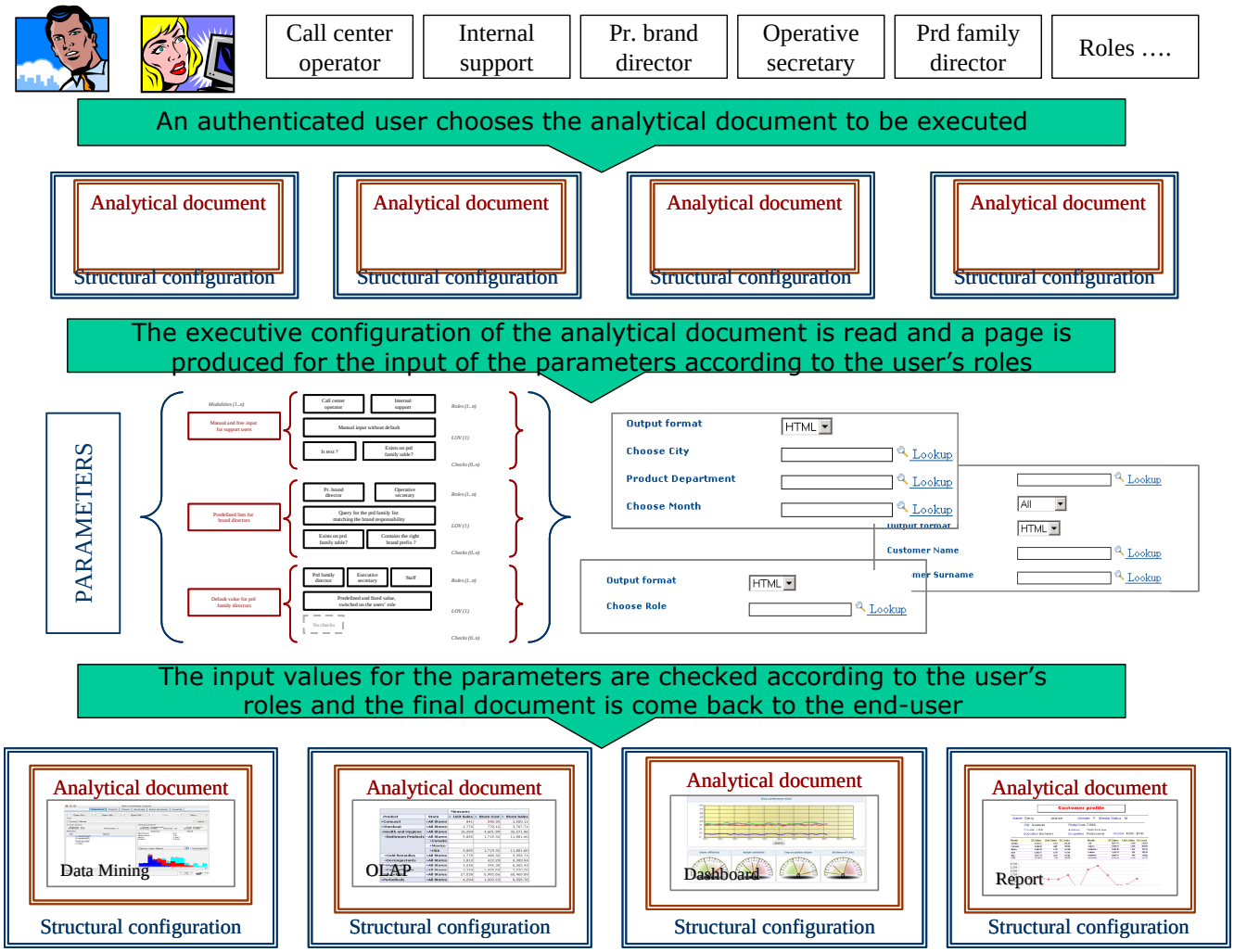



**Fig. 6 – Analytical document execution**

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

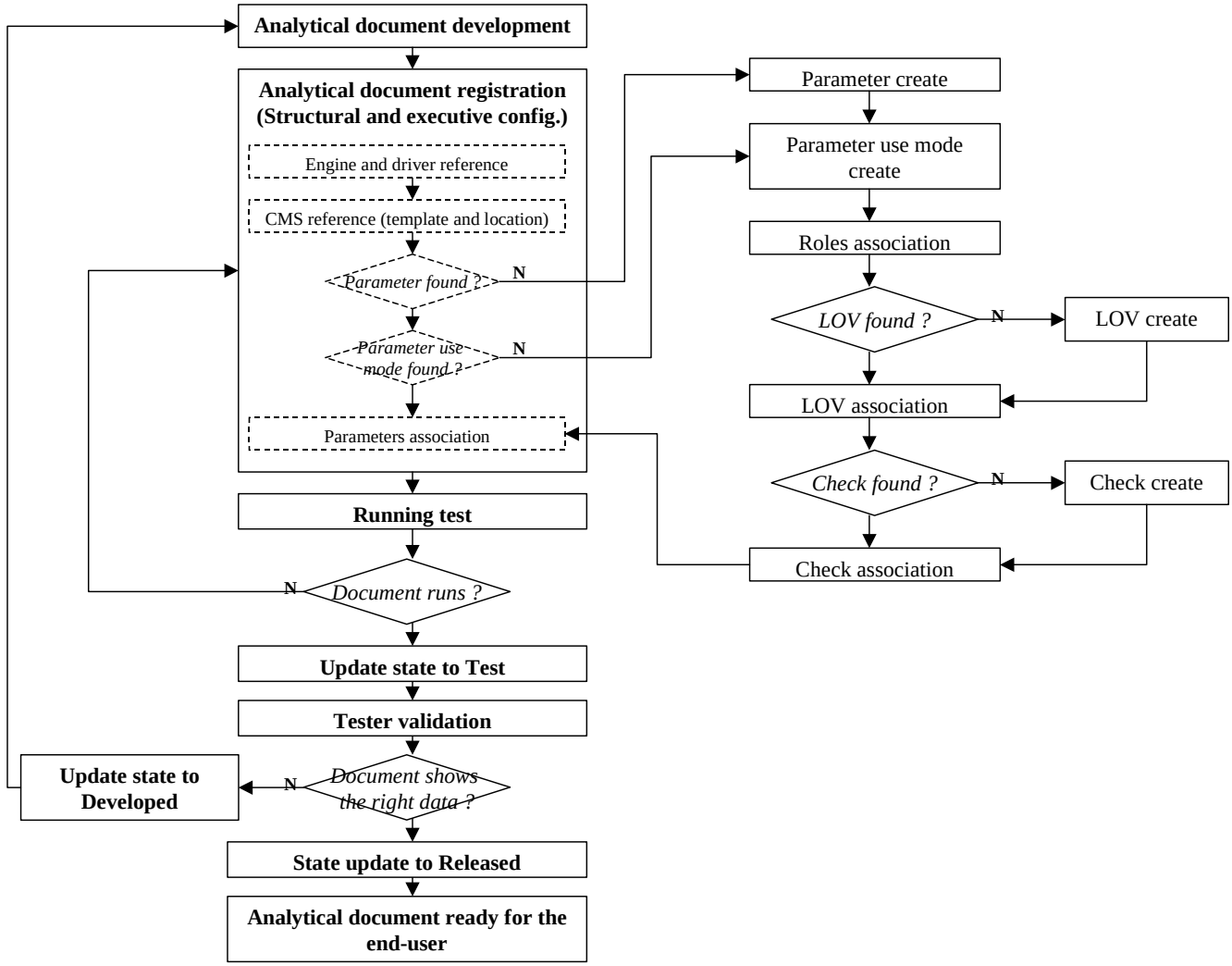



**Fig. 7 – Analytical document handling**

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.

# 3 Recurring themes

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

## 3.1 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.

## 3.2 List and detailed View

One of the most common views 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.

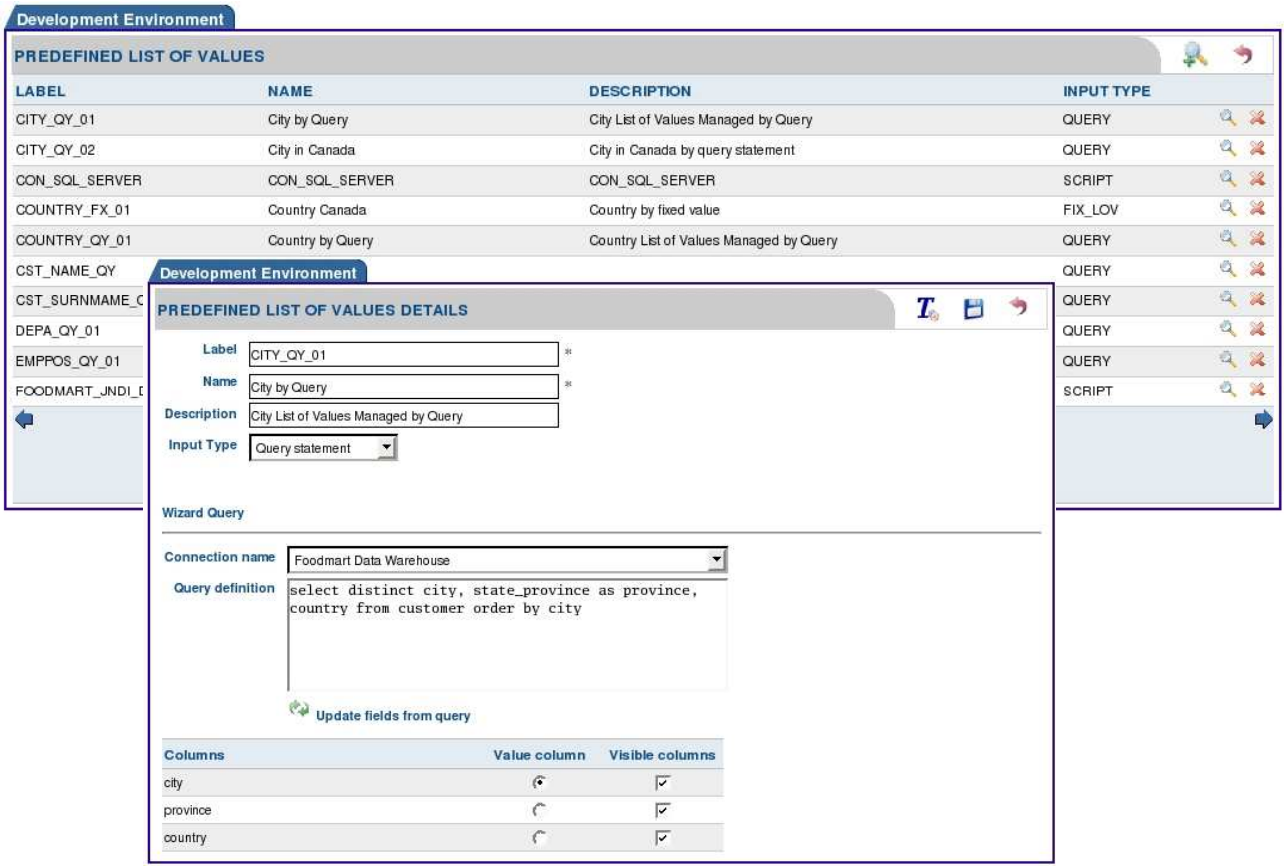



**Fig. 8 – List-details Example**

# 4 Functionality overview

## 4.1 ADMINISTRATOR

☺ 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.

The administrator (biadmin/biadmin user) main tasks are:

- Registering and configuring each analytical engine used inside the platform;
- Configuring the functional structure that classifies the analytical documents and distributes the rights required in order to use it and to access it;
- Maintaining the registered analytical documents.

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

These functions are provided by means of a single portlet that can be included into a portal environment supporting its specification.

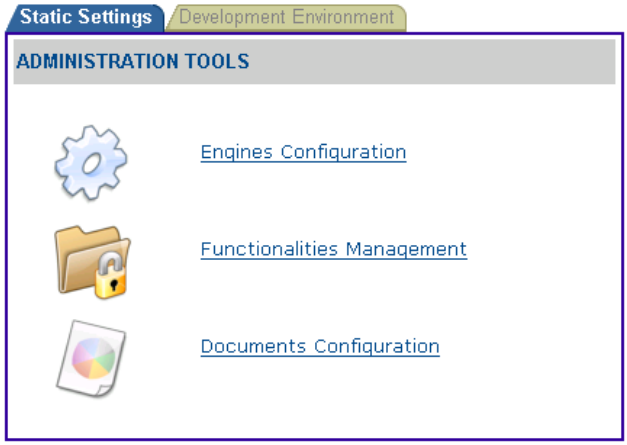



**Fig. 9 – Administrator portlet**

### 4.1.1 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*.

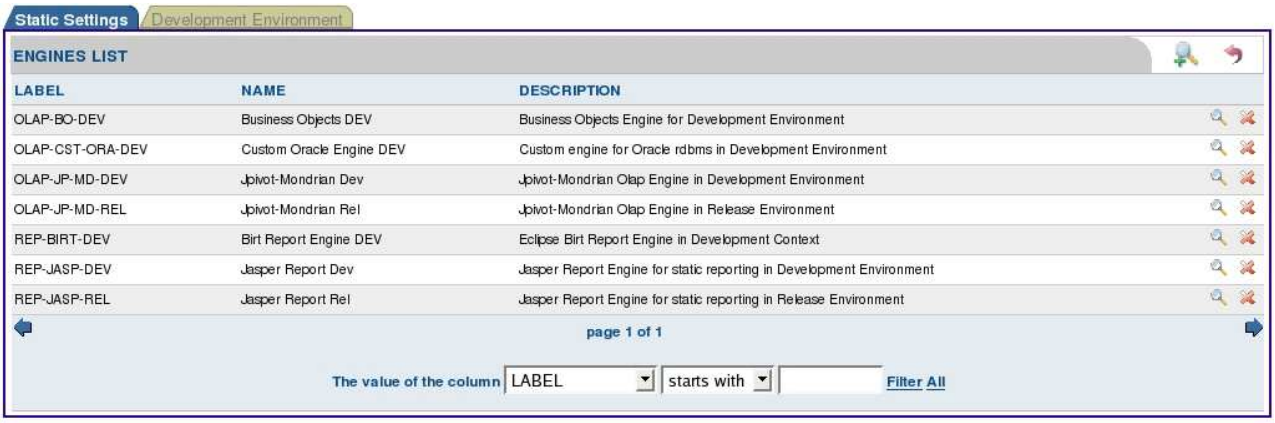



**Fig. 10 – Engines list**

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

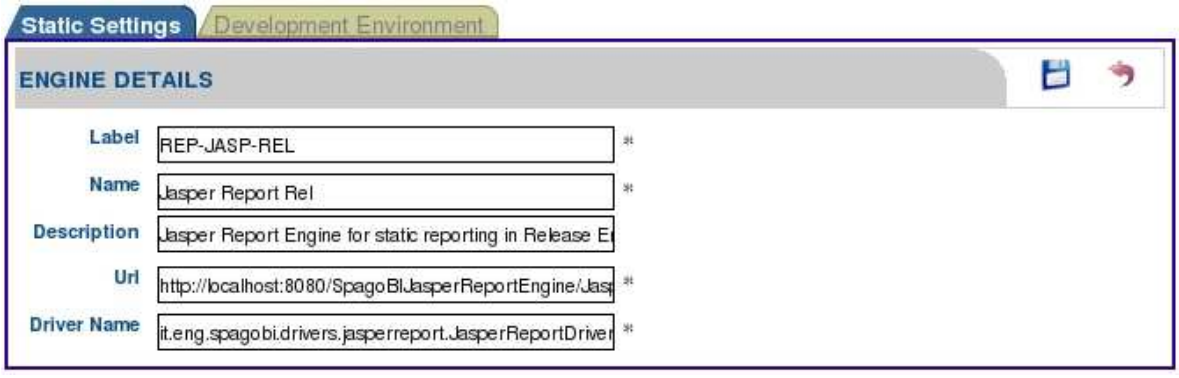



**Fig. 11 – Engine details**

The current SpagoBI version uses the following drivers:

- *JasperReport*: report analysis;
- *Jpivot*: OLAP analyis;

### 4.1.2 FUNCTIONALITIES MANAGEMENT

SpagoBI uses its own file system, named "**Functionality Tree**", that allows to better organize documents by grouping them by folder regulating the access to them.
This multi-level hierarchical structure (Fig. 12) can be created and modified exclusively by the administrator in the "Function Administration" area.

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

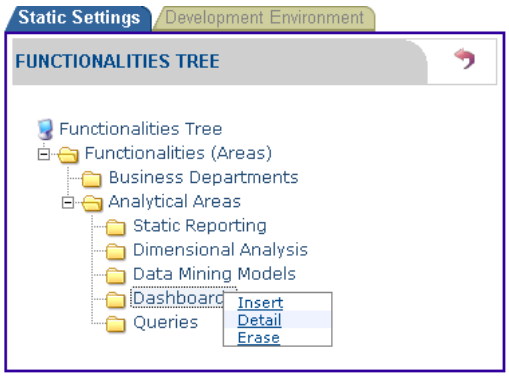



**Fig. 12 – Functionalities Tree**

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.

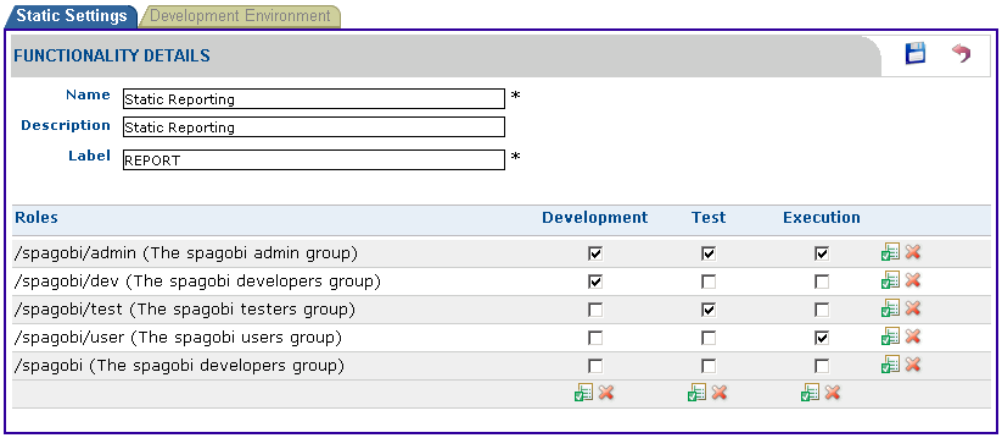



**Fig. 13 – Functionality details**

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 of each role.

For instance, checking the Development and the Test boxes of the "SpagoBI user group", all the users logged as "biuser" will only be able to develop and test documents belonging to the selected functionality, but not to execute them.

Notice that every new node inherits (by default) all its father's rights.

### 4.1.3 DOCUMENT CONFIGURATION

A list of all the *Analytical Documents* registered in SpagoBI can be listed accessing to the *Document 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.

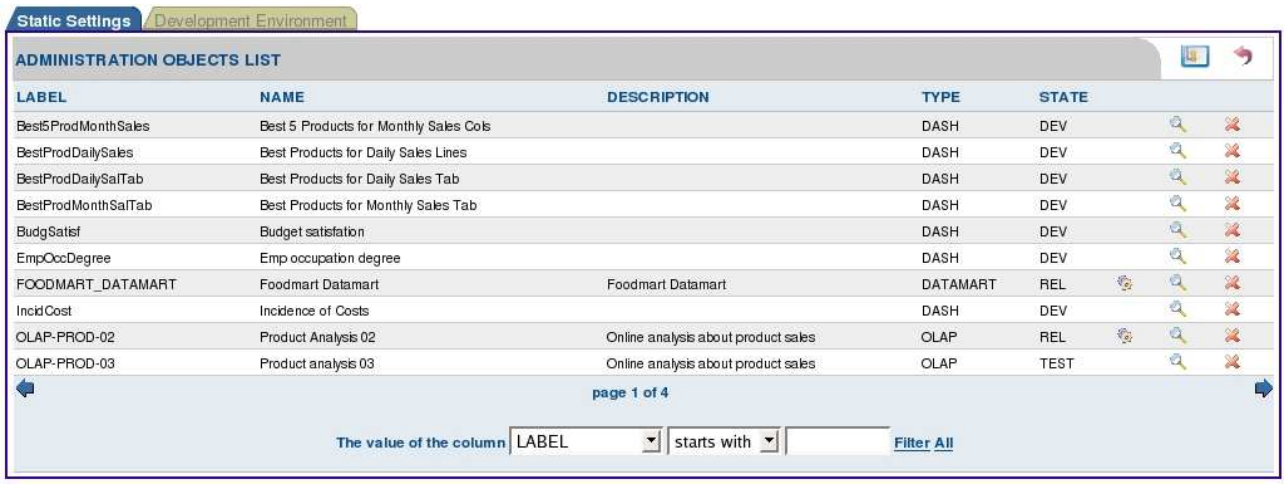



**Fig. 14 – Administration Objects List**

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.

The administrator has the *List View* but he can switch to the *Functionality Tree* in order to have a list of documents grouped by functionality, clicking on the VIEW AS TREE icon.

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

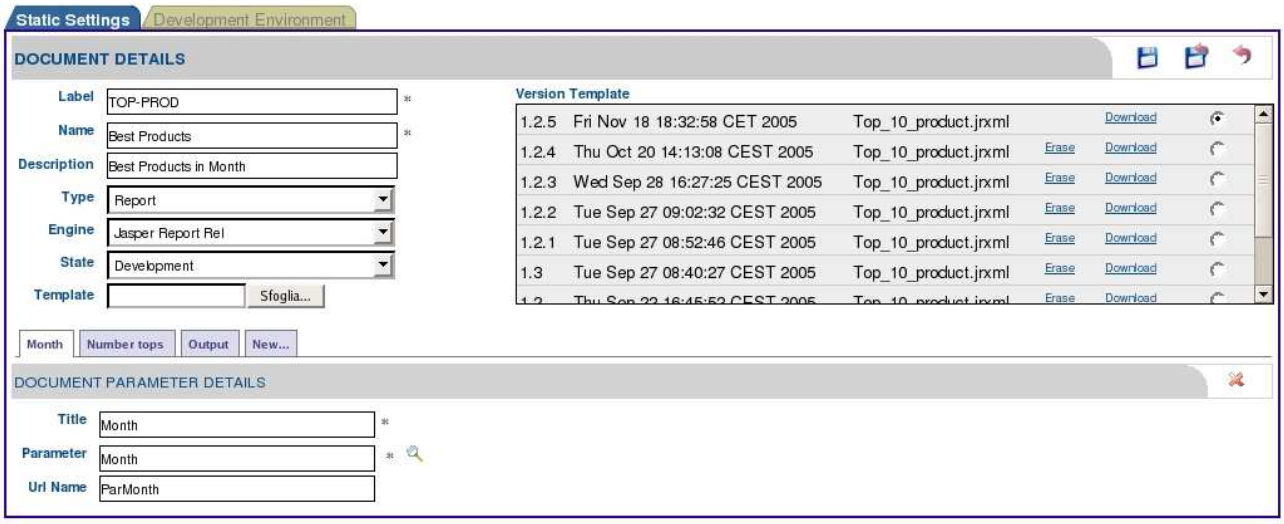



**Fig. 15 – Administration Object Details**

- **Engine**: the name of a registered engine that has to be used to execute the *Analytical Document*;
- **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 has been 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.

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 name 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.

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.

## 4.2 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 each analytical document, referring to the used parameters.


**Fig. 16 – Developer tools**

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.

### 4.2.1 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.

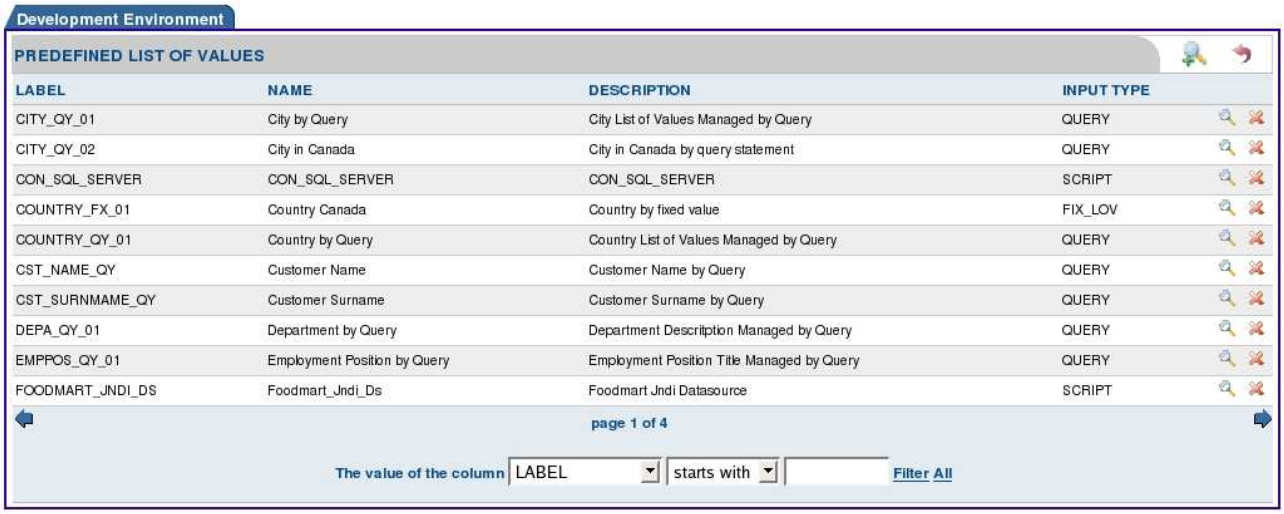



**Fig. 17 – Predefined List of Values**

Accessing the details page, general information are displayed:

- **Label**: LOV unique identifiers;
- **Name**: LOV name;
- **Description**: brief description of the LOV (optional);
- **Input Type:** four different types of LOV are admitted allowing:
  - **Manual Input:** the free parameters allocation;
  - **Query:** the database retrieval of all the selected values;
  - **Fixed LOV**: the organization of an arbitrary value list;
  - **Script:** the registration of methods and classes delegated to the recovering of all the values to suggest.

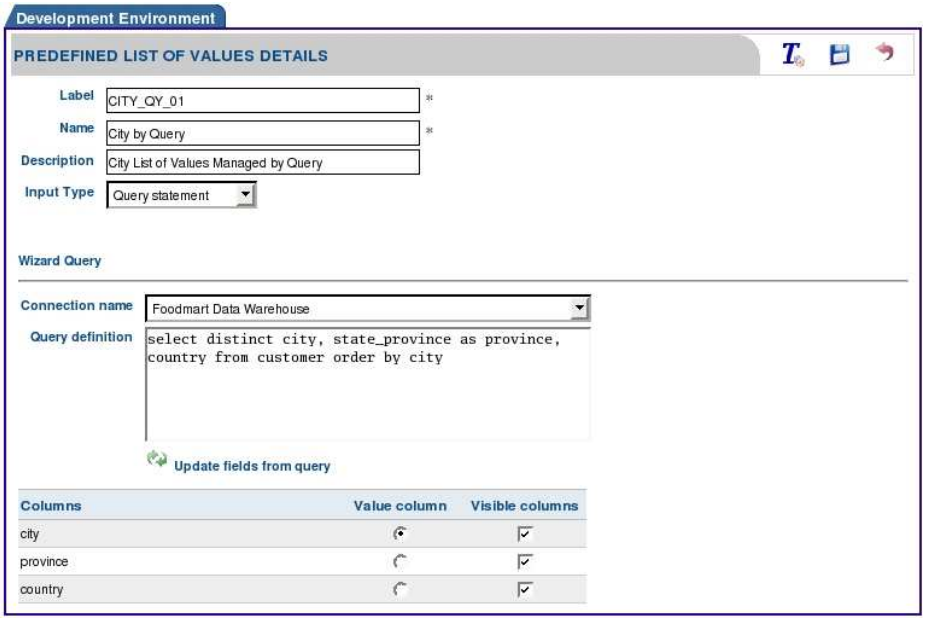



**Figure 18 - Predefined List of Values Details**

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.

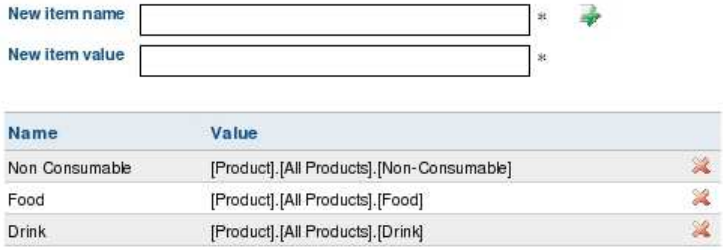



**Figure 19 – Fixed values wizard**

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

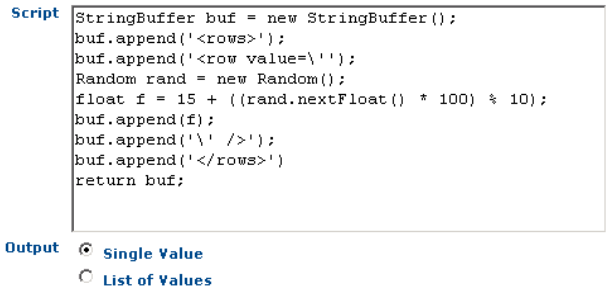



**Figure 20 – Script wizard**

### 4.2.2 PREDEFINED VALUES CONSTRAINTS

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

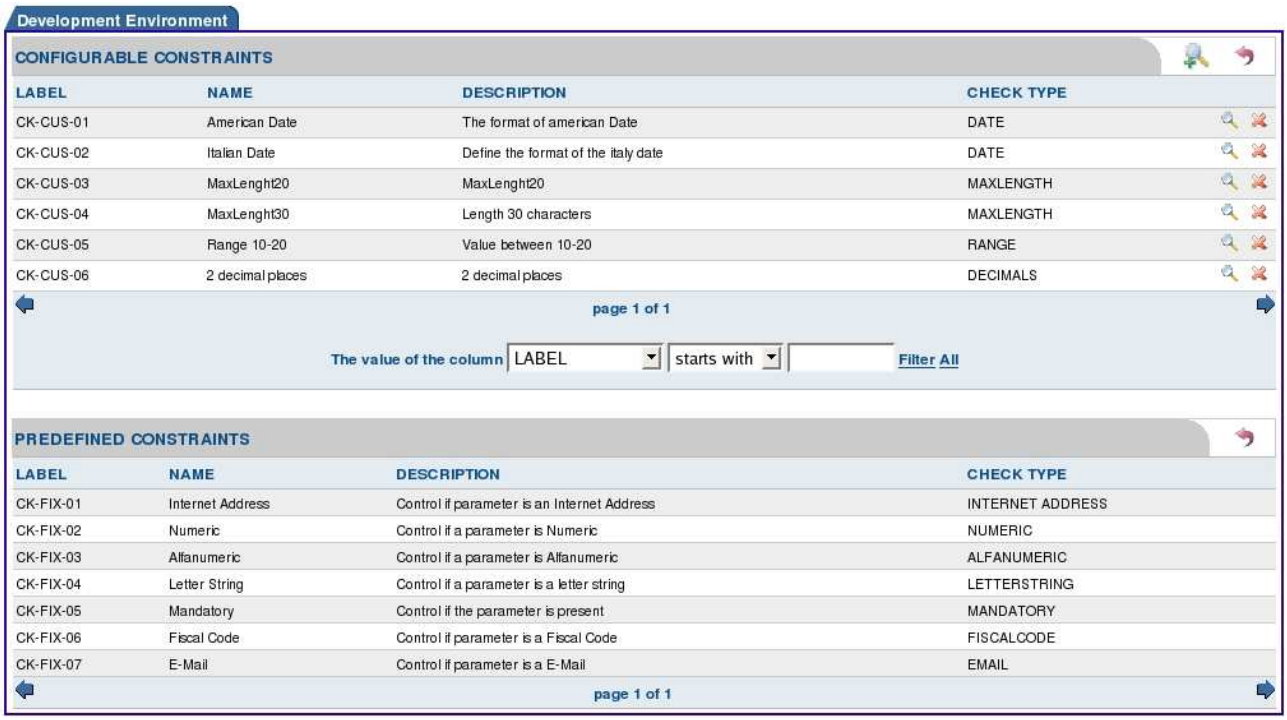



**Figure 21 - Configurable Constraints and Predefined Constraints**

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 22 - Constraints 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*.

### 4.2.3 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.

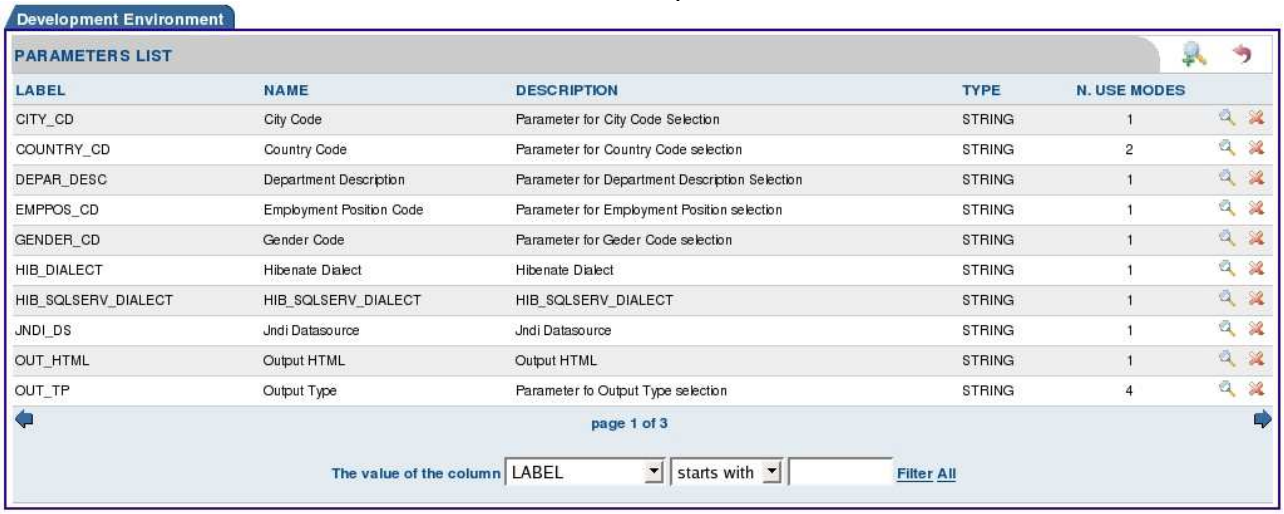



**Figure 23 – Parameters list**

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;

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

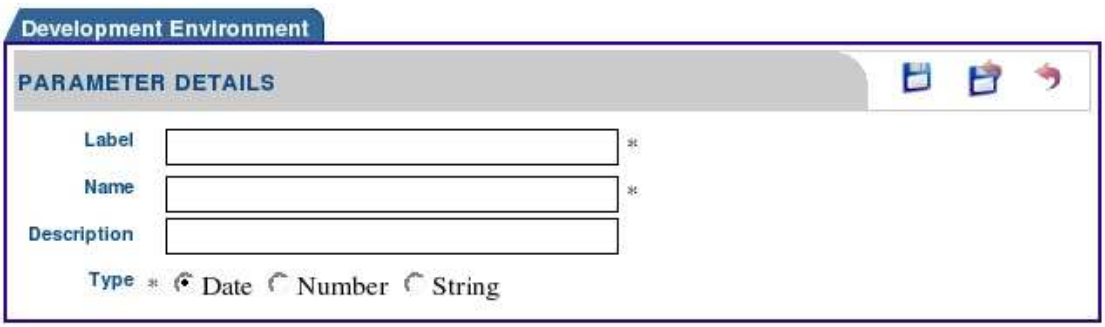


**Figure 24 – New parameter form**

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):

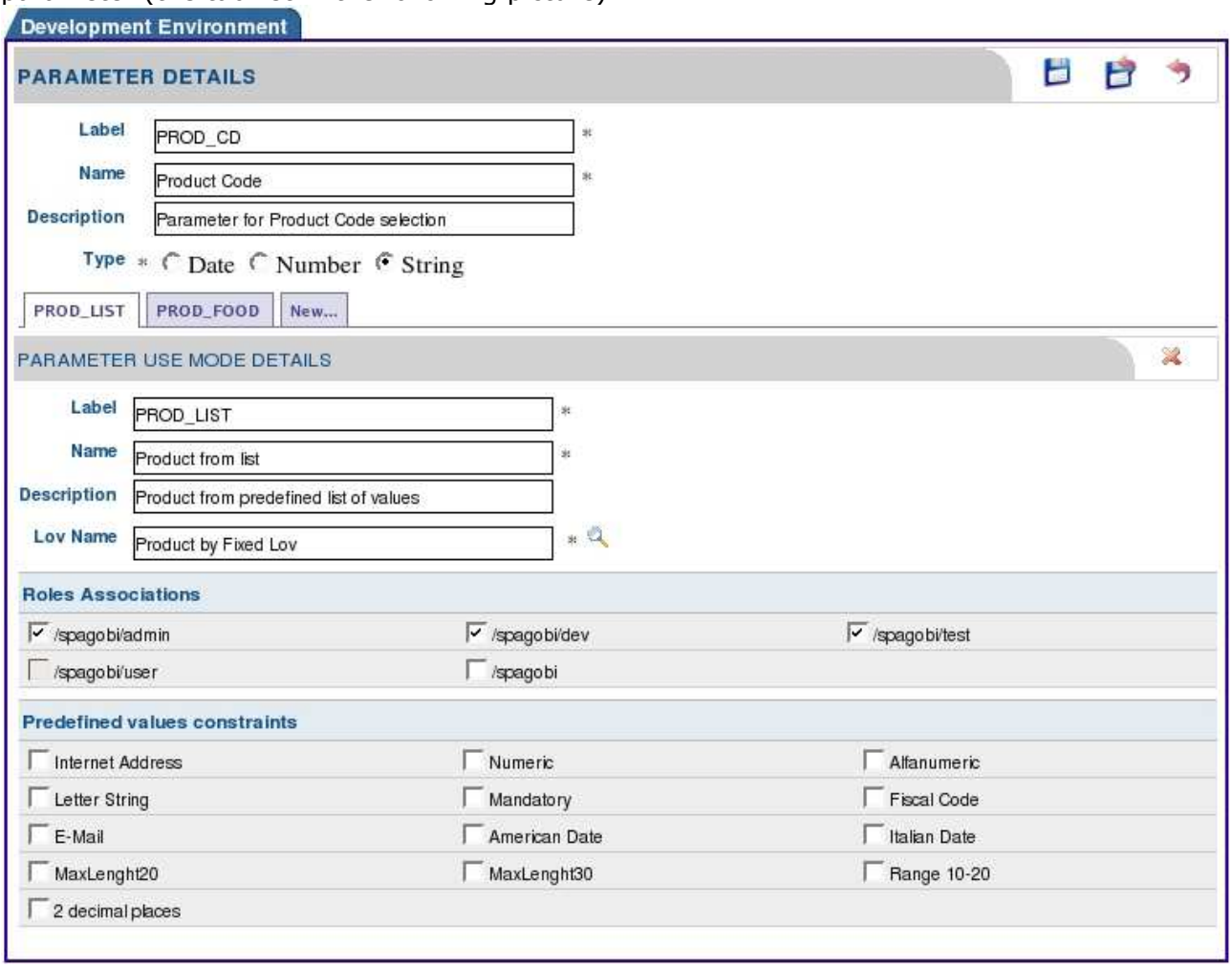



**Figure 25 – Parameter details**

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*;
- **Predefined List of Value**: 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;
- **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 would not be associated to any

Parameter *Use Mode*. In the second case a user having just that role will not be able to use a document related to this Parameter;

- **Predefined Values Contraints:** it is possible to assign zero, one or more constraints selecting them from the list of available constraints. To create a new *Predefined Values Contraints* please refer to the Predefined Values Constraints paragraph.

### 4.2.4 DOCUMENT CONFIGURATION

The user, simply by clicking on *Document 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.).

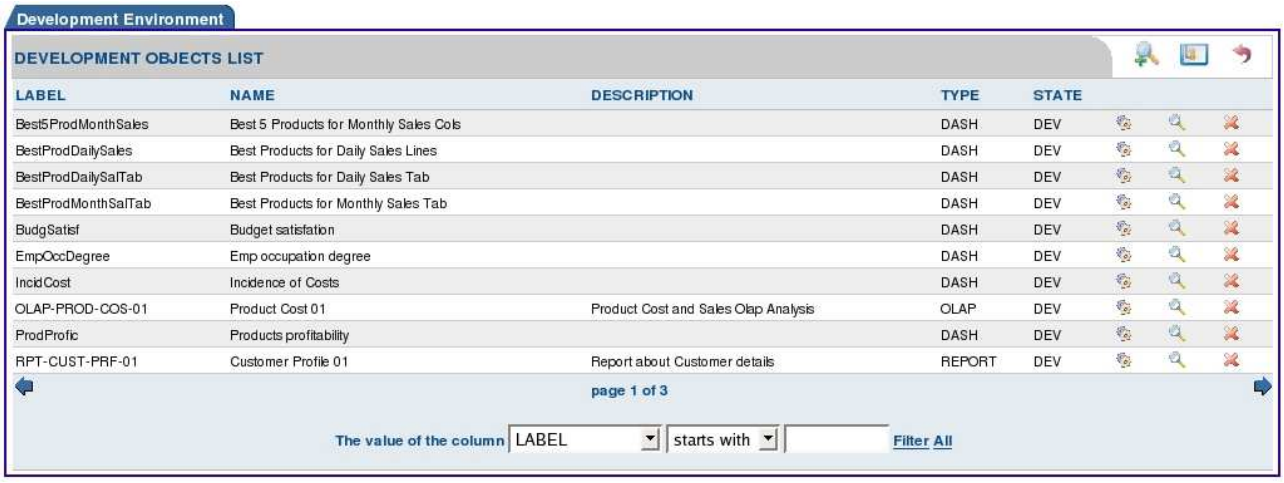



**Figure 26 – Analytical documents list**

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);
- **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.

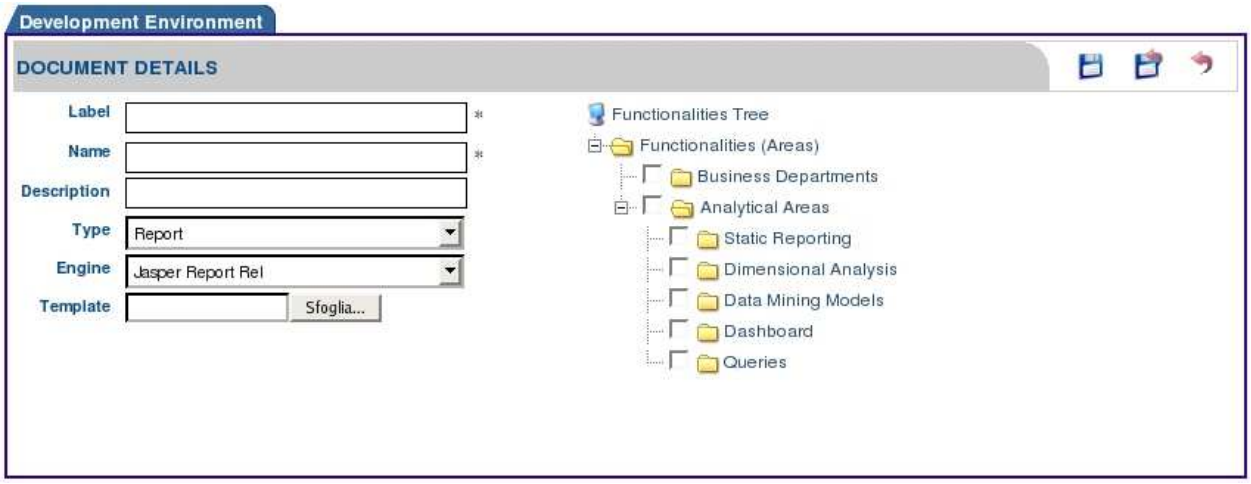



**Figure 27 – Analytical document details (new document)**

Multiple selection is allowed but at the moment is not implemented.

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): 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;
- **Parameters tabs** (bottom side): there is one tab for each parameter associated at the Analytical document. An additional tab (*New …*) creating a new association is provided.

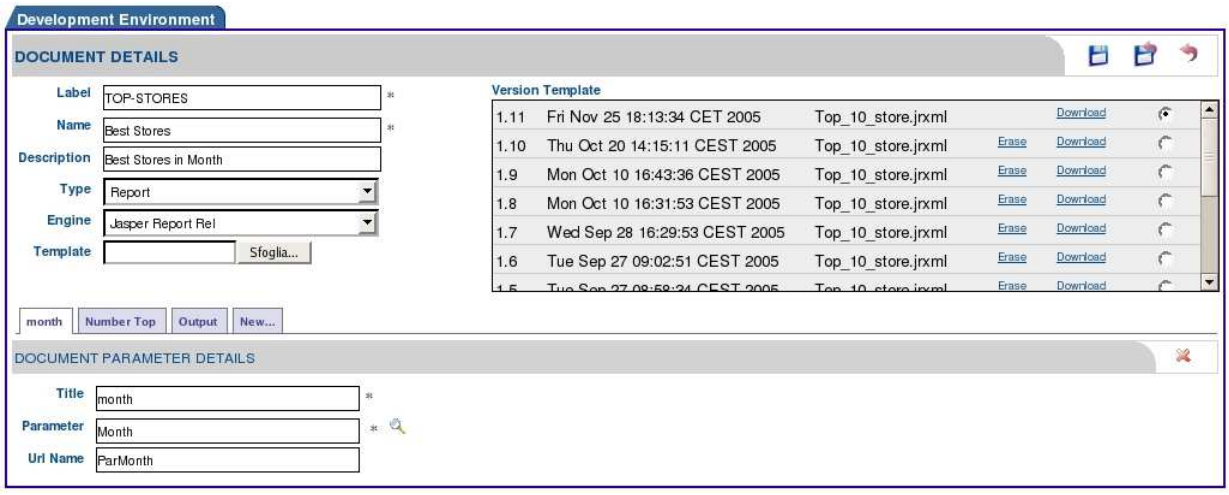


**Figure 28 – Analytical document details (existing document)**

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.

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.

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.

## 4.3 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.

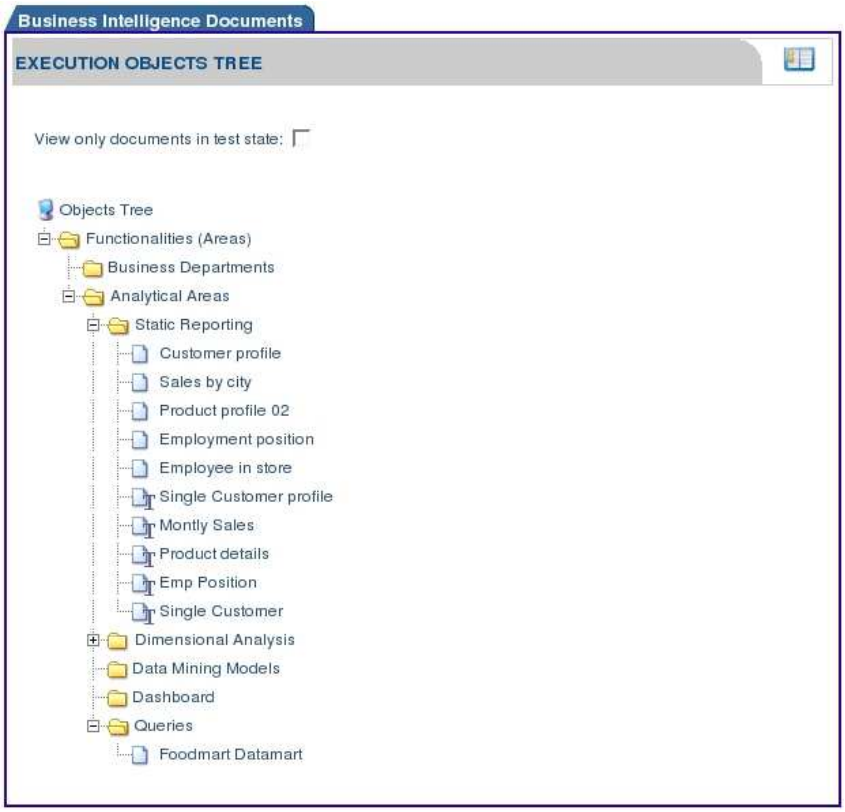


**Figure 29 – Tester's documents tree**

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 shows a "T" in its icon). 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.

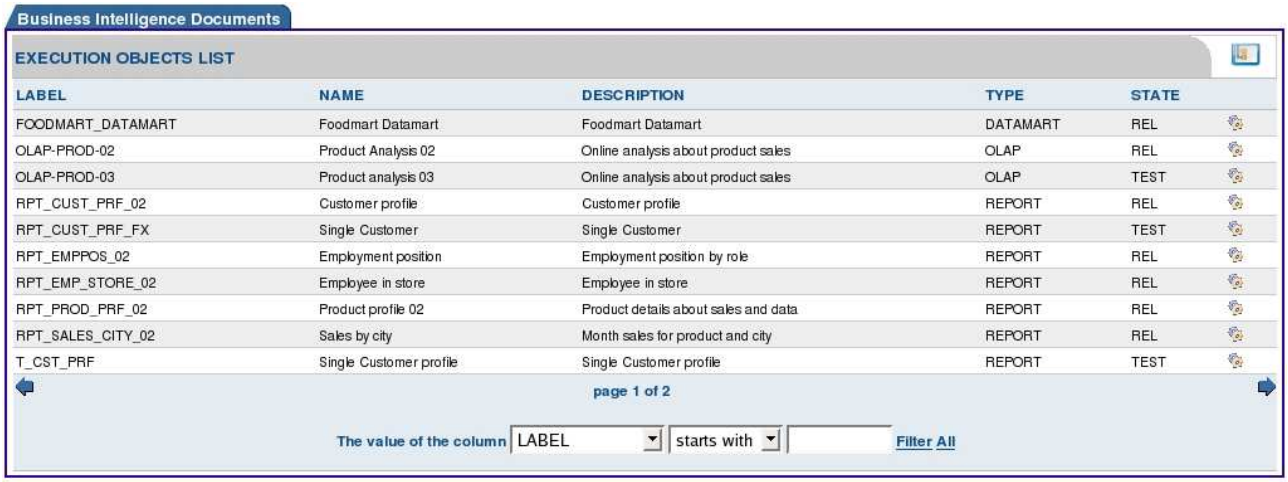



**Figure 30 – Tester's documents list**

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.

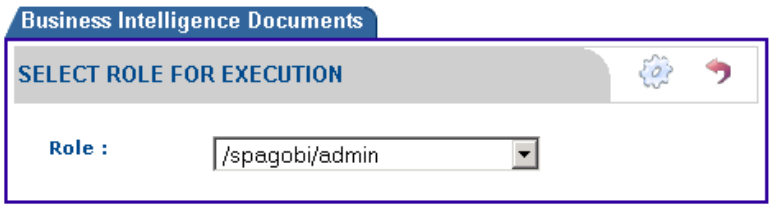



**Figure 31 – Role selection page**

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).

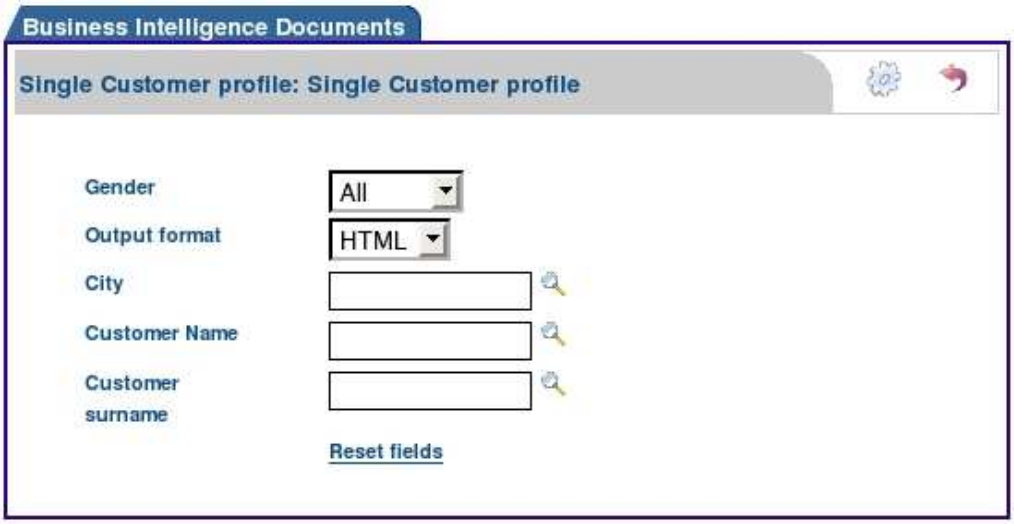



**Figure 32 – Parameters page**

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.

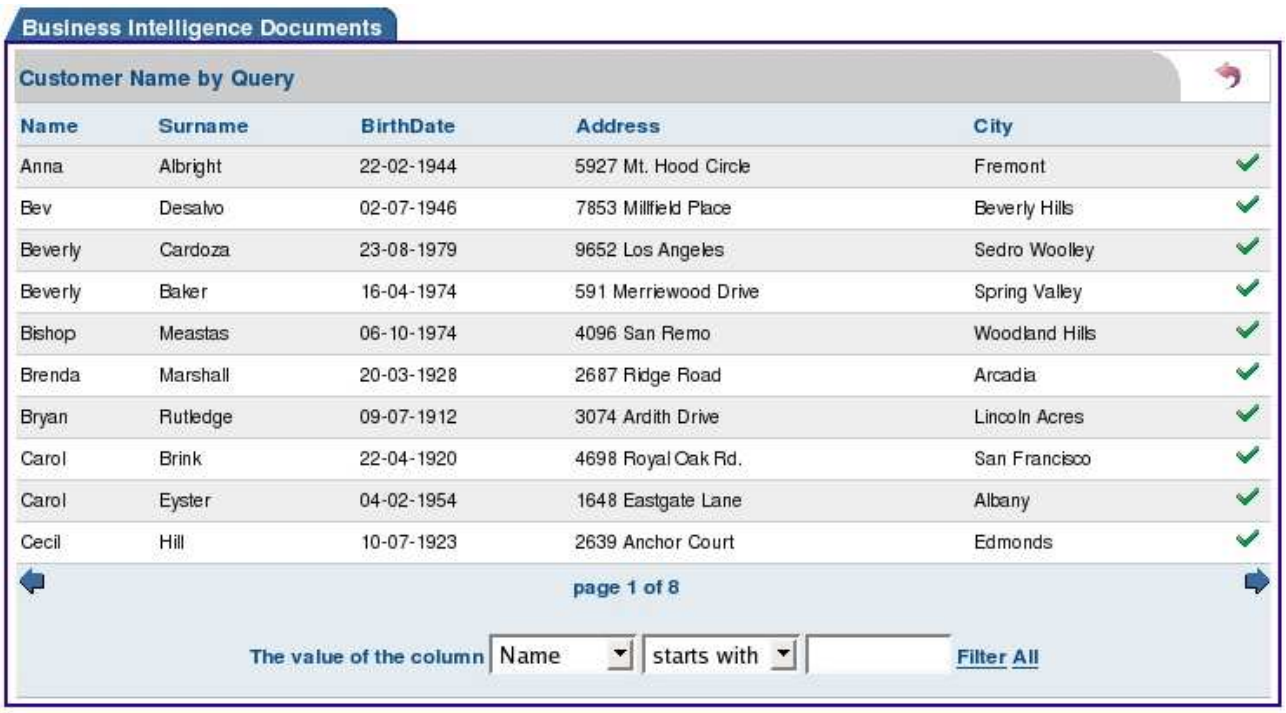


**Figure 33 – Lookup for parameter input value**

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.

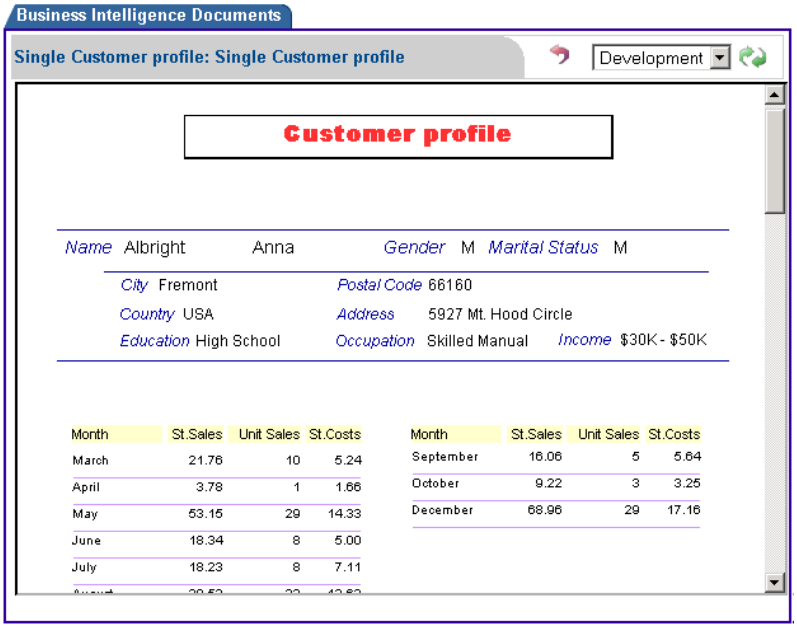



**Figure 34 – Final execution**

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.

## 4.4 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.

### 4.4.1 QbE: 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 35 – 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 seven 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 HSQL language;
- **Save Query**: to verify if the query is formally correct and to save it;
- **Result Preview**: to display a preview of the results obtained by the query realized.

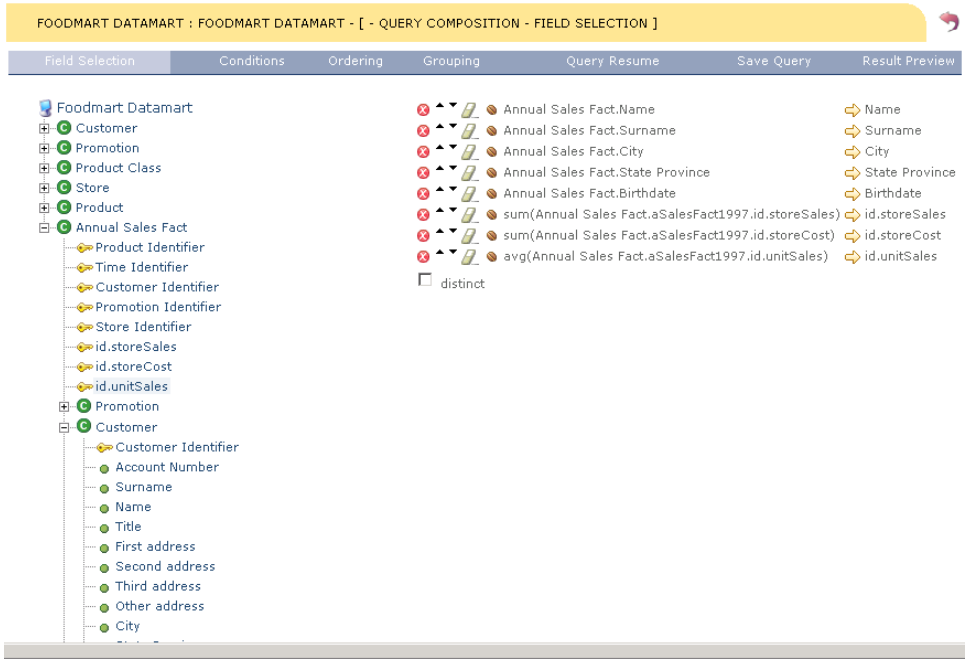
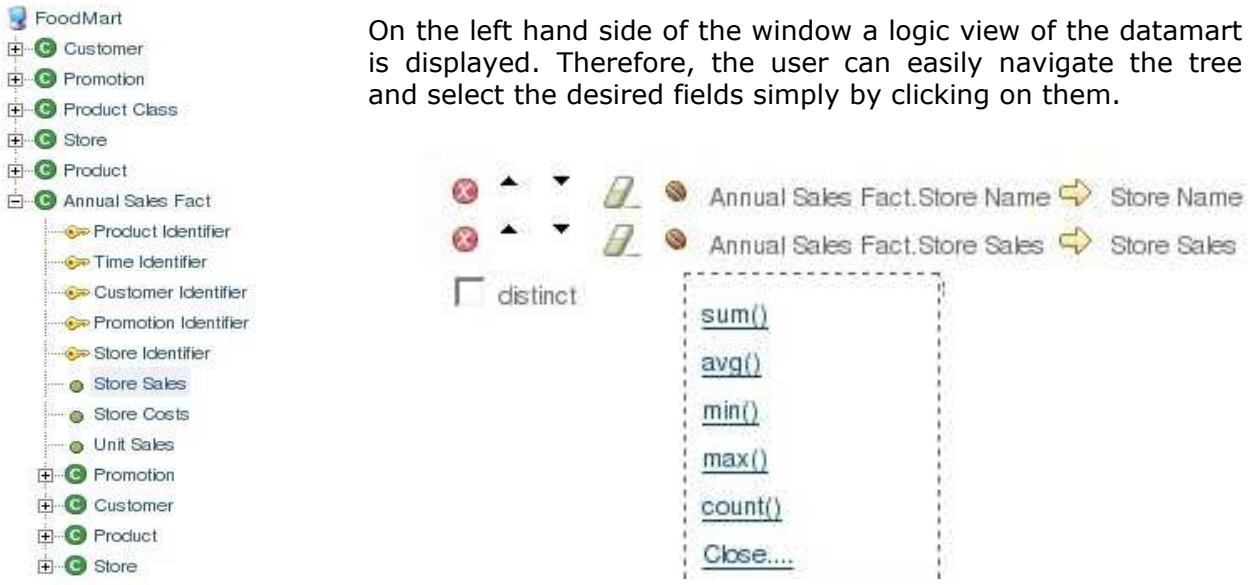



**Figure 36 – QbE-Field Selection**

The user can display each section simply by clicking on the corresponding tab on the title line at the top of the window.
In the following paragraphs each section will be described in more detail.

#### 4.4.1.1 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 37 – 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.

### 4.4.1.2 Condition

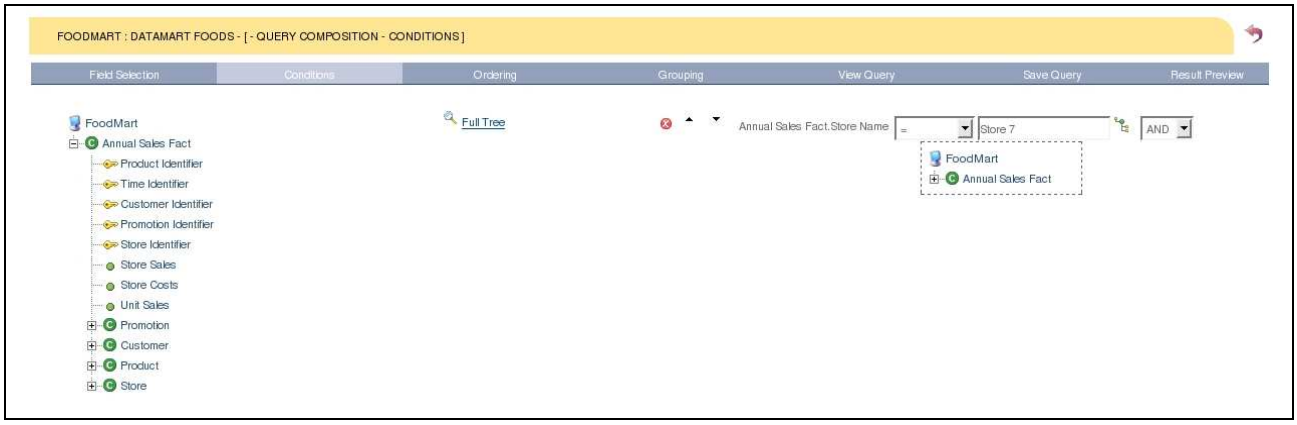


**Figure 38 – QbE- Condition**

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 39 – 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

### 4.4.1.3 View Query

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

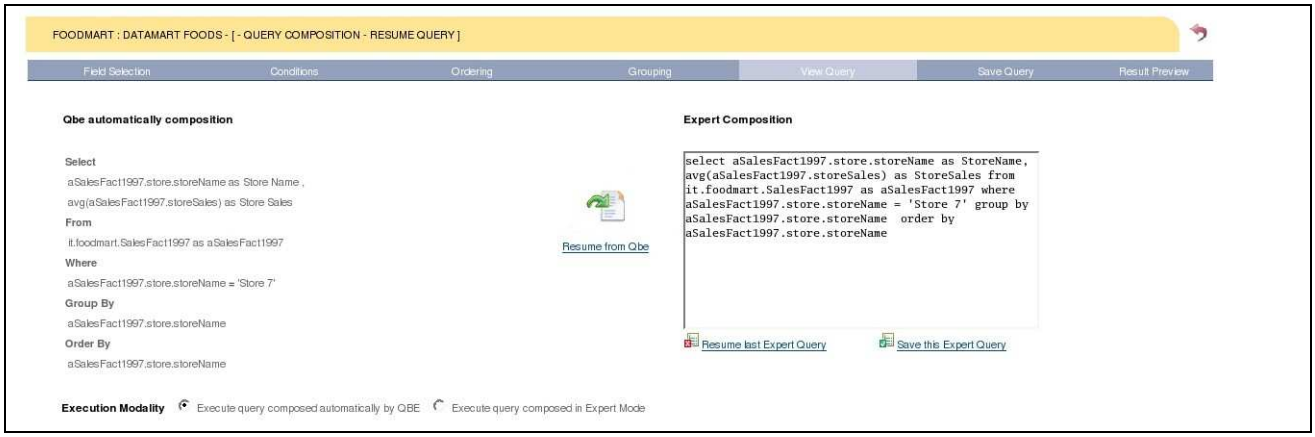



**Figure 40 – QbE – View Query**

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.

### 4.4.1.4 Save Query

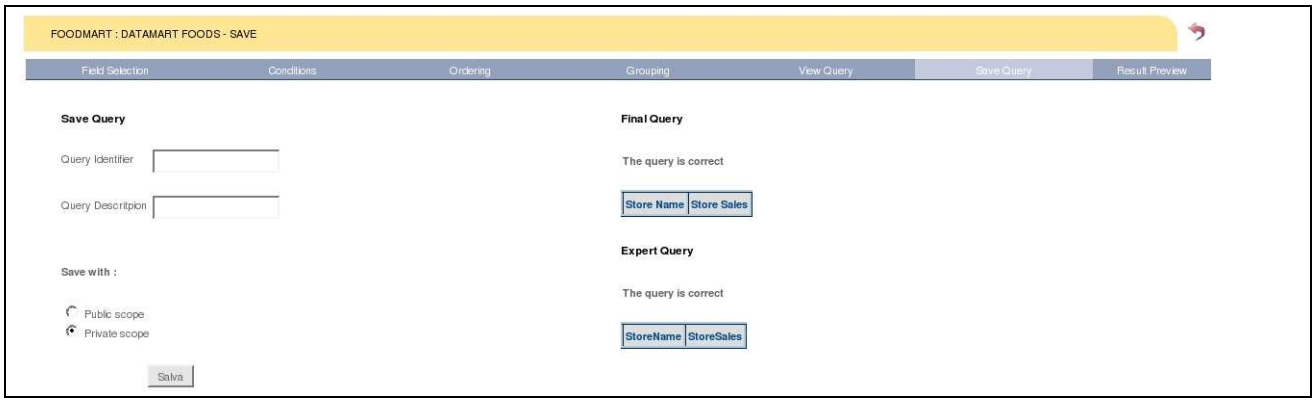


**Figure 41 – QbE – Save Query**

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 be visible to
- Private scope: the query will be visible to

#### 4.4.1.5 Result Preview

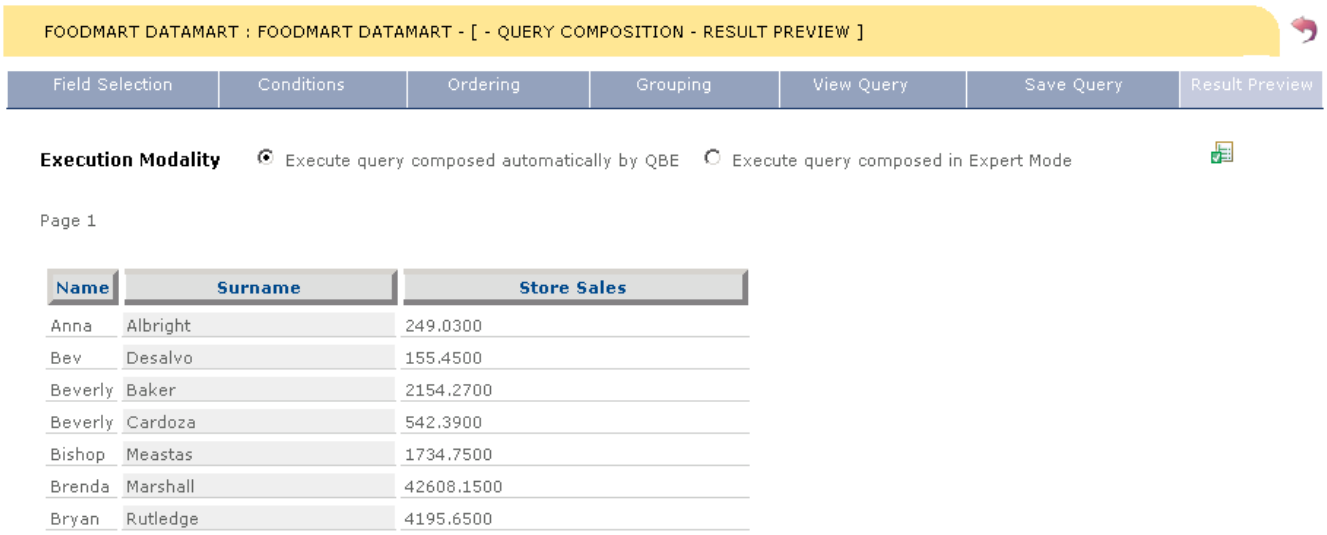



**Figure 42 – QbE – Result preview**

The **Result Preview** 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.

# 5 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.

## 5.1 Install ExoTomcat and SpagoBI DEMO

Download ExoTomcat 1.0 from :
http://forge.objectweb.org/project/showfiles.php?group_id=151&release_id=1378

Unzip the file exoplatform-tomcat-1.0.zip but do NOT run the application, because this would cause the failure of the following SpagoBI installation.

Download SpagoBI Demo 1.7 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 1.0.

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*

## 5.2 Report

The main steps to manage a report are:

1. Create a Report Template using IReport
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 exist and the developer can use what he prefers in order to produce the report template.

### 5.2.1 CREATE A REPORT TEMPLATE USING IREPORT

1. If you don't have iReport 0.5.1 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".

### 5.2.2 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.

#### 5.2.2.1 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.

#### 5.2.2.2 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 .

#### 5.2.2.3 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*.

### 5.2.3 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
- **Template**: click on the *browse* button to select the report template created in the Create a Report template using IReport 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 using IReport 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 document.

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

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

### 5.2.4 TEST THE ANALYTICAL DOCUMENT

Work in progress.

### 5.2.5 EXECUTE THE ANALYTICAL DOCUMENT

Work in progress.

## 5.3 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

### 5.3.1 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;
- <connection> logical name that will be interpreted by the engine to identify the proper connection ;
- <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>
  <connection name='defaultDWH' />
  <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*.

### 5.3.2 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.

#### 5.3.2.1 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.

#### 5.3.2.2 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.

#### 5.3.2.3 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*.

### 5.3.3 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.

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**: 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.

9. 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.

10. Click on the icon EXECUTE which corresponds to the new *OLAP - Document*.
11. In the new page you will be required to enter a value for the *OLAP - Doc Param*.
12. Select *Food* from the combo-box.
13. Click on the EXECUTE icon to execute the document.
14. 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.
15. Logout

### 5.3.4 TEST THE ANALYTICAL DOCUMENT

Work in progress.

### 5.3.5 EXECUTE THE ANALYTICAL DOCUMENT

Work in progress.

## 5.4 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

### 5.4.1 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.

### 5.4.2 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="spagobi/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="/spagobi/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.

### 5.4.3 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

#### 5.4.4 TEST THE ANALYTICAL DOCUMENT

Work in progress.

#### 5.4.5 EXECUTE THE ANALYTICAL DOCUMENT

Work in progress.

### 5.5 DATA MINING

Work in progress.

### 5.6 QUERY BY EXAMPLE

Work in progress.

# 6 In more depth

### 6.1 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.

## 6.2 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.

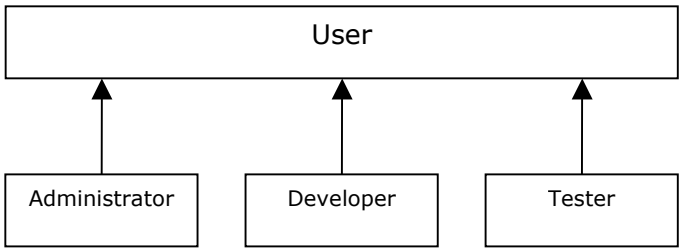



**Figure 43 User type hierarchy**

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.

## 6.3 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.

## 6.4 Document organization and Security policy

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

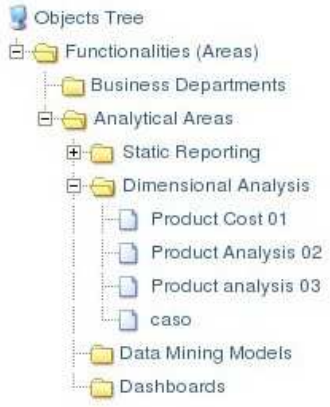



**Figure 44 Functionalities Tree**

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.

## 6.5 User definition and roles management

Work in progress.

## 6.6 Portal definition

Work in progress.

## 6.7 Add an Enging

Work in progress.

## 6.8 Functionality tree management

Work in progress.

## 6.9 Data Mart (.jar) development for QbE Feature

### 6.9.1 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 and to inquire it with SpagoBI it's necessary to define a new **datamart** object. This kind of object doesn't need an engine and as template it takes the jar containing the hibernate class and mapping files.
Since qbe is based on hibernate it needs also to know the kind of the target database in order to use the right sql dialect and to connect to it. To satisfy the two previous requirement it's mandatory to define two parameters for each datamart object:

- one parameter with url_name equals to '**JNDI_DS**' which must be filled with the jndi name of a datasource for the target database. Qbe will use it to get the connection for the inquiry
- one parameter with url_name equals to '**DIALECT**' which must be filled with the complete hibernate dialect class name suitable for the target database. Qbe will use it to choose the right sql dialect for the inquiry

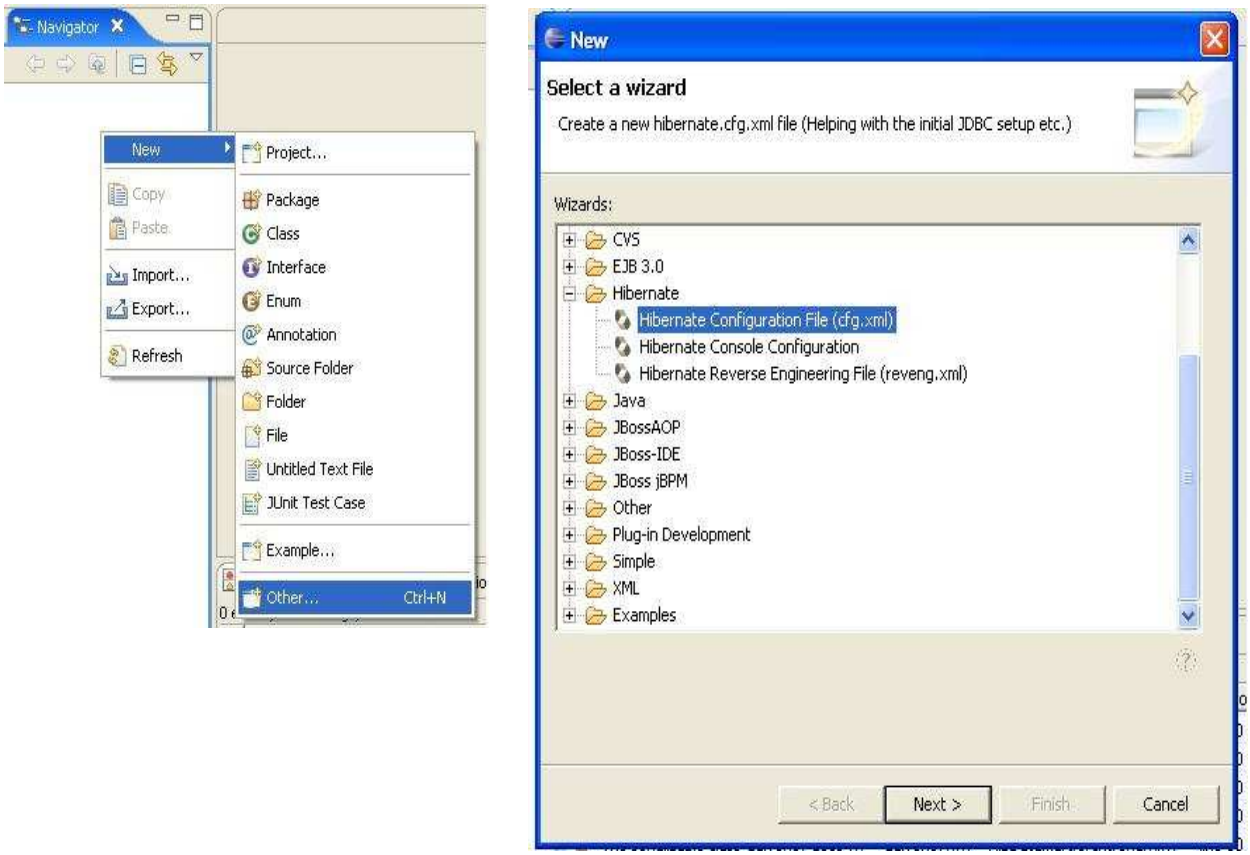
### 6.9.2 How to create database model jar file

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:

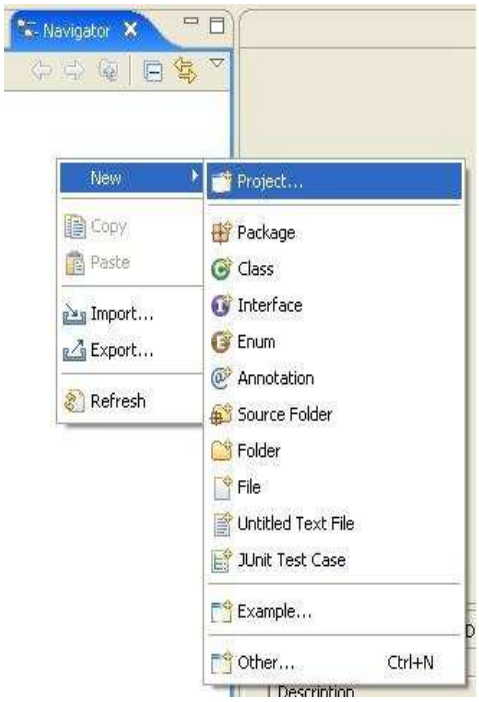
- download eclipse 3.1 sdk. It's strongly recommended to install the tools over a new and clean installation of eclipse 3.1. It's possible to download the 3.1 version from the url http://download.eclipse.org/eclipse/downloads/index.php
- install the eclipse platform: simply unzip the zipped downloaded file everywhere you want; a new folder named eclipse will be created (ECLIPSE-HOME)

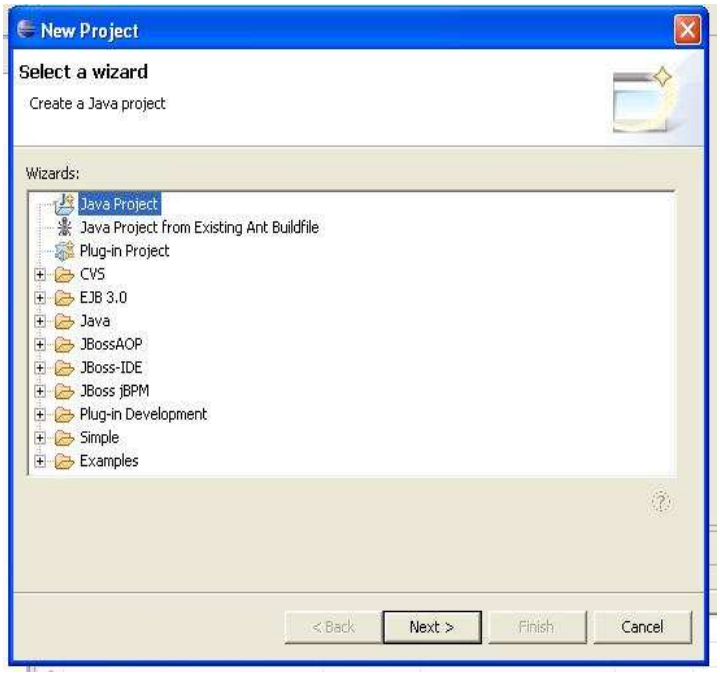
- download JbossIDE 1.5 final tools file(which contains the hibernate tools). It's possible to download it selecting the JBossIDE-1.5-ALL.zip file from the link http://sourceforge.net/project/showfiles.php?group_id=22866&package_id=72248&release_id=382687
- install the JBoss IDE tools over the eclipse 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. (Figures 1 - 2)




**Figures 1 – 2 Eclipse Navigator and New 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 figures 3 - 4 – 5.




**Figures 3 – 4 new java project**

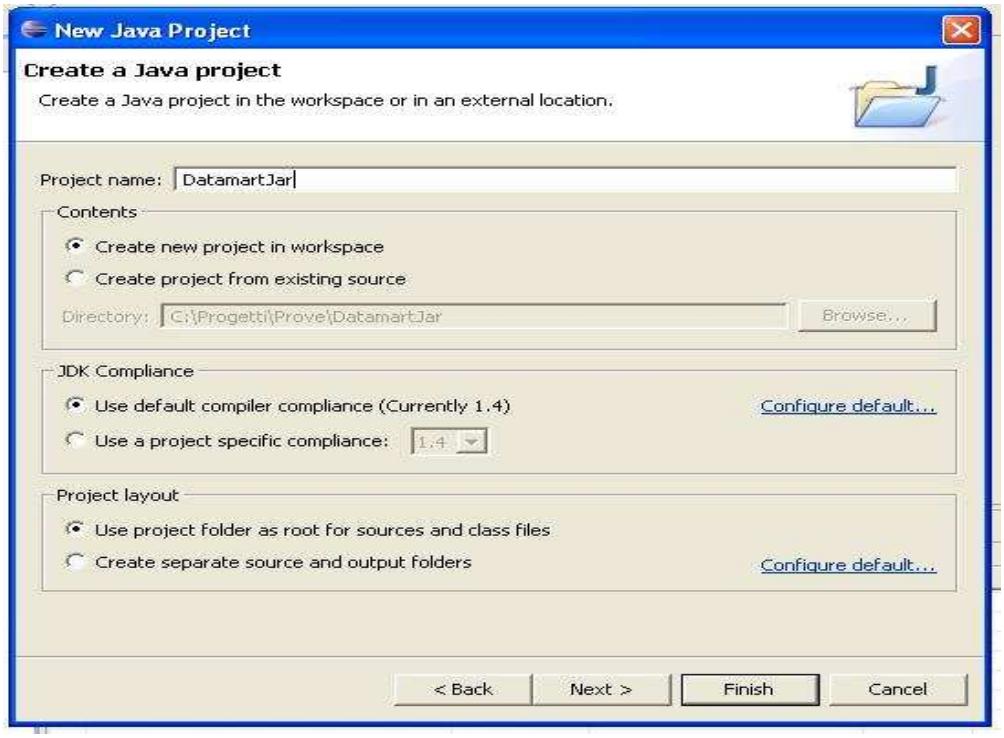



**Figure 5 Default properties of a java project**

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 6)

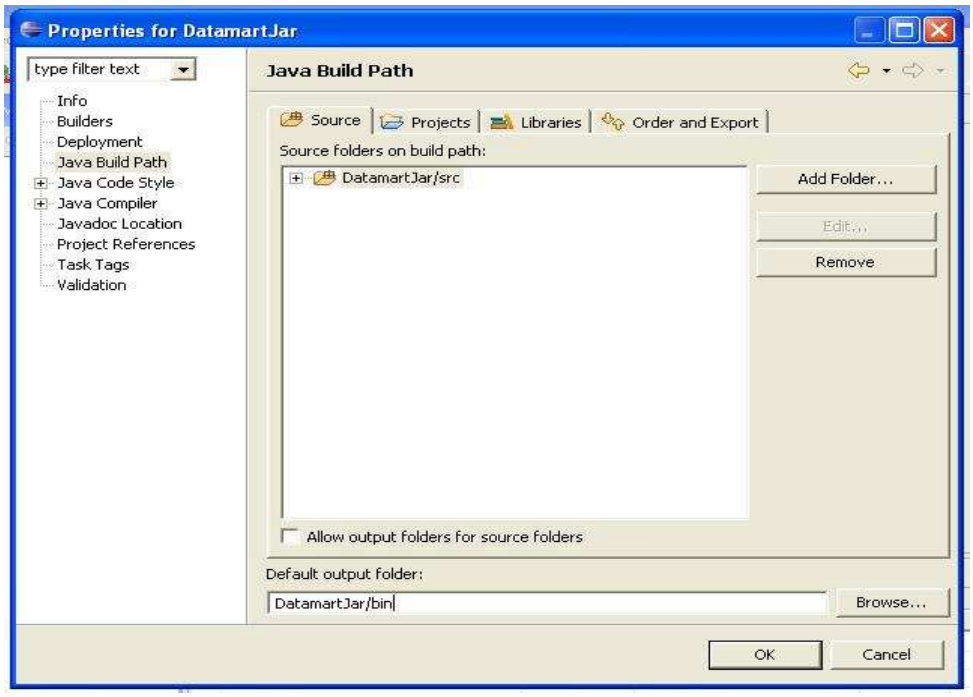



**Figure 6 Java project build path options**

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. (Figures 7-8)

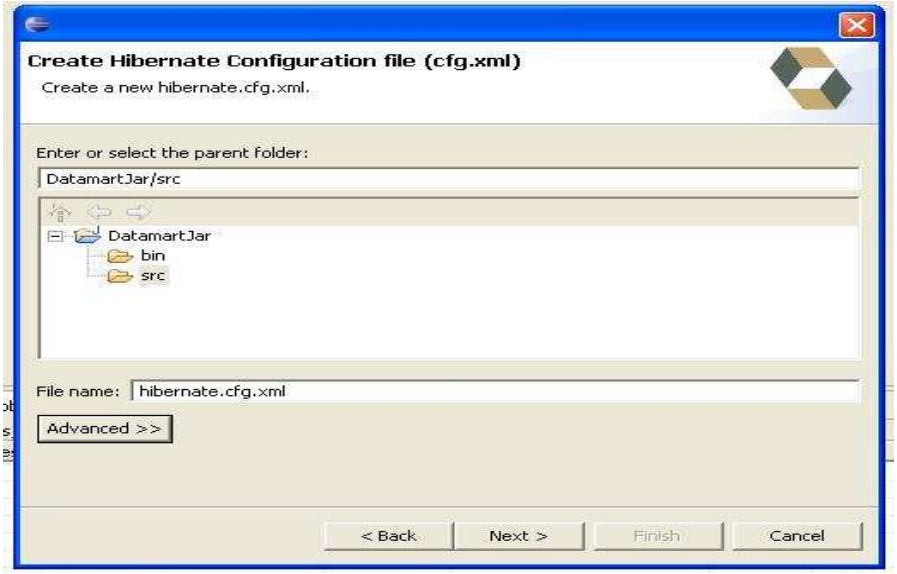



**Figure 7 New Hibernate configuration file**

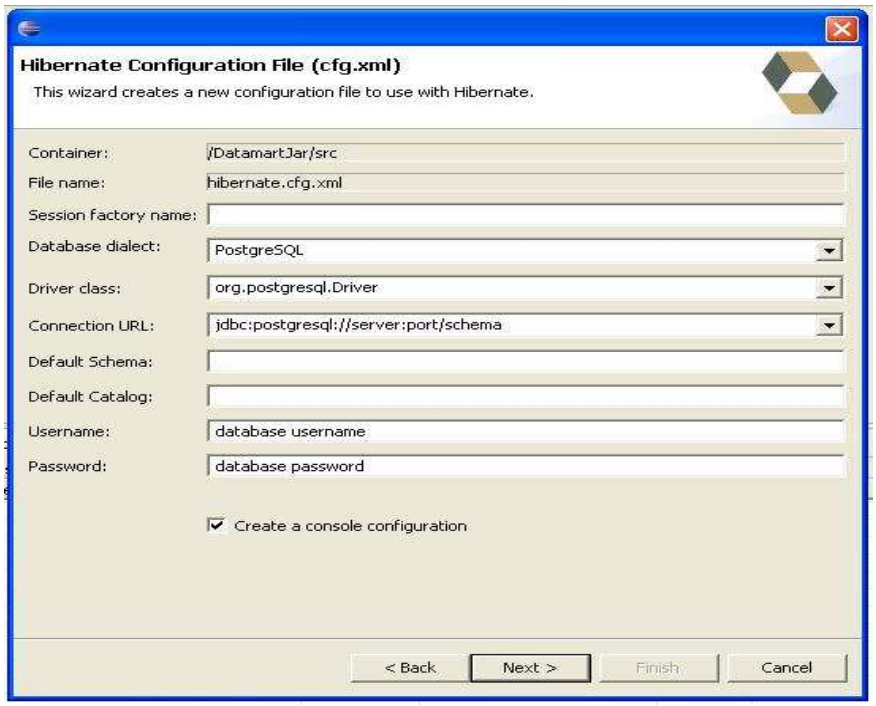



**Figure 8 Hibernate configuration properties**

Give a name to the console, add the jar of the database driver into the classpath and then press the finish button (Figure 9)

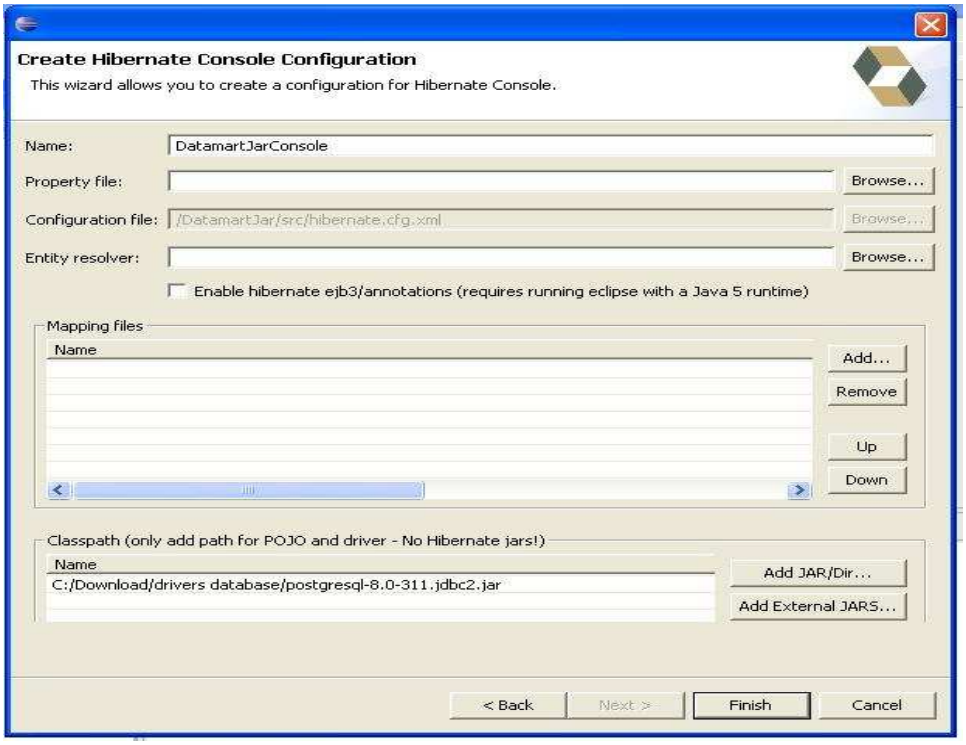



**Figure 9 New Hibernate console**

Exec the hibernate code generation tool (Figure 10)


**Figure 10 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 11)

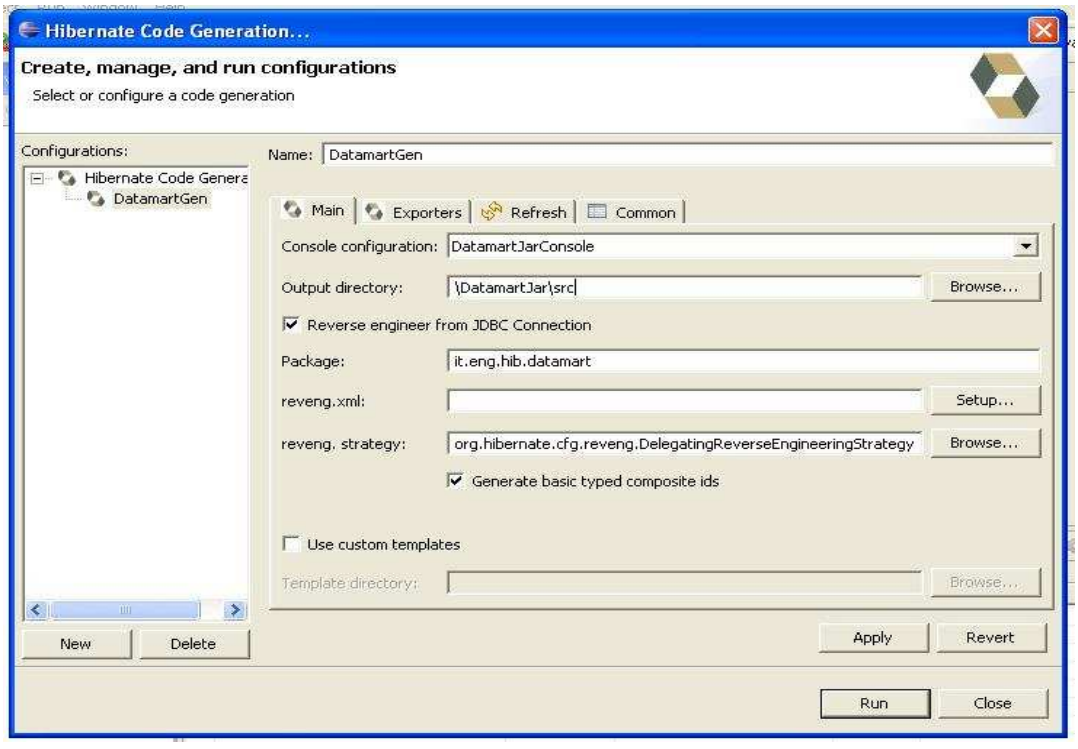



**Figure 11 Hibernate code generation main options**

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 three options as in the figure 12.

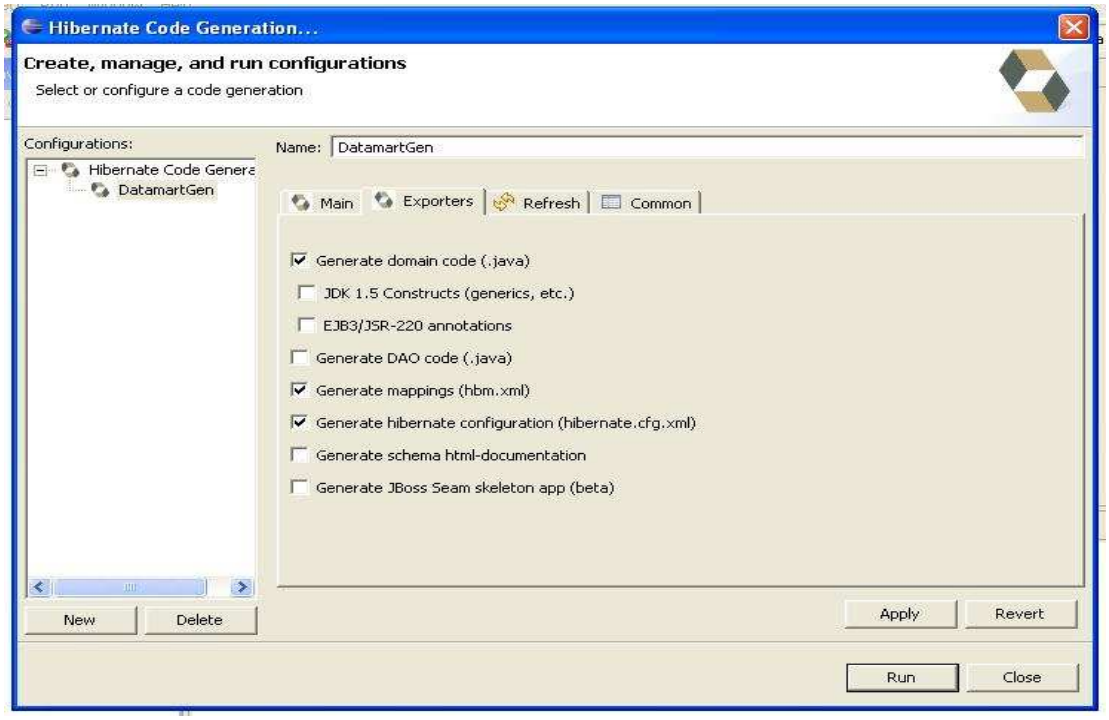



**Figure 12 Hibernate code generation exporters options**

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 13.
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.

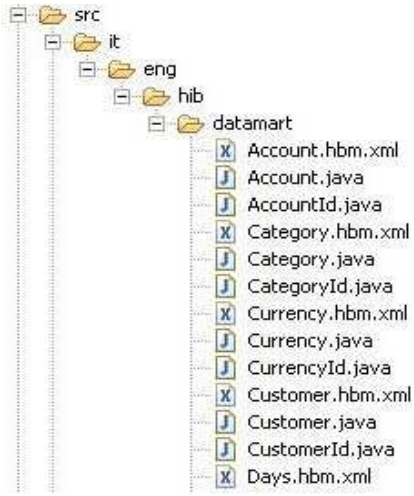



**Figure 13 Java classes and Hibernate mapping files**

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 14)

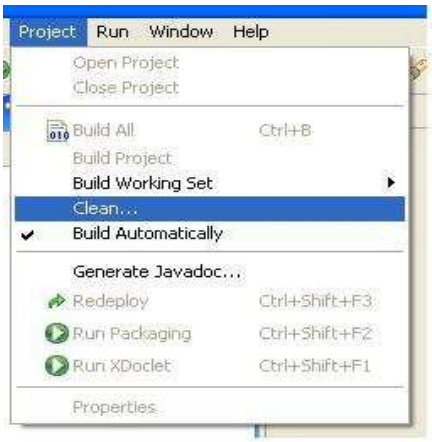



**Figure 14 Compile project**

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. (Figures 15 - 16)

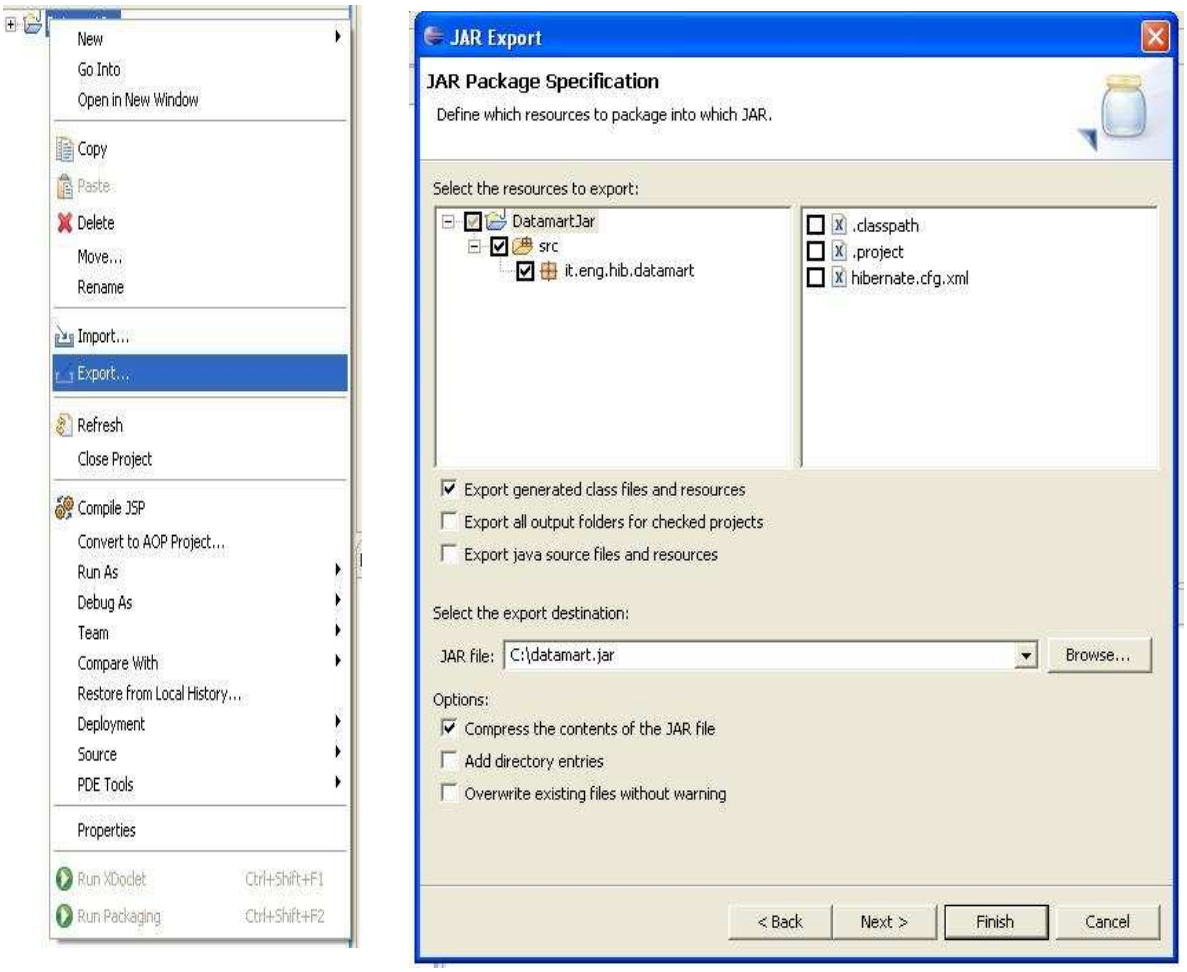



**Figure 15 – 16 Export jar file**

The datamart jar model can be taken from the destination directory and loaded as template into a SpagoBI datamart Object.

# 7 Glossary

Work in progress.

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