BASI DI DATI 2

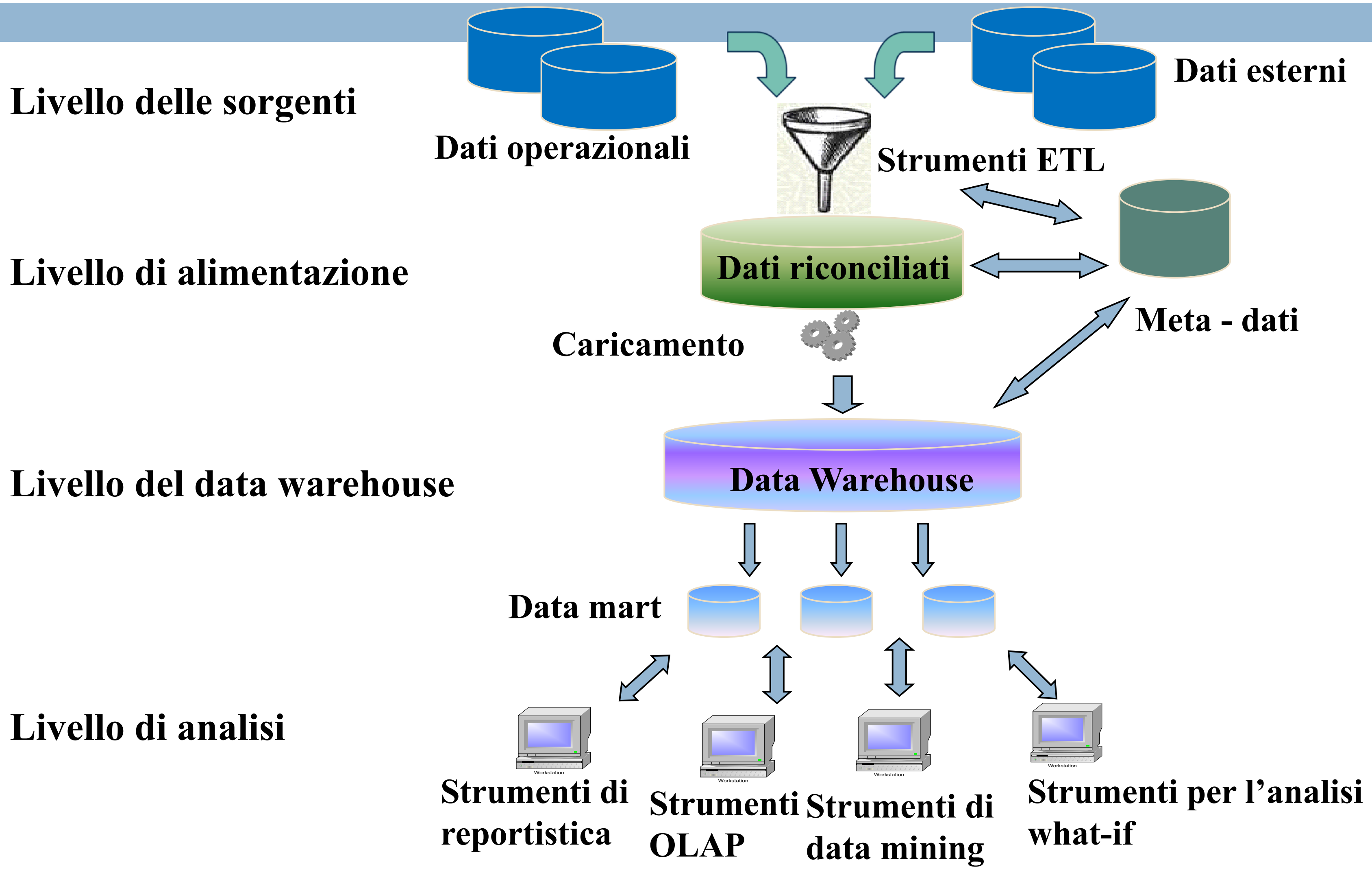
TUTORIAL. *PENTAHO SOLUTIONS:*

ETL – IL TOOL KETTLE
SCHEMA WORKBENCH
BI SERVER & JPIVOT

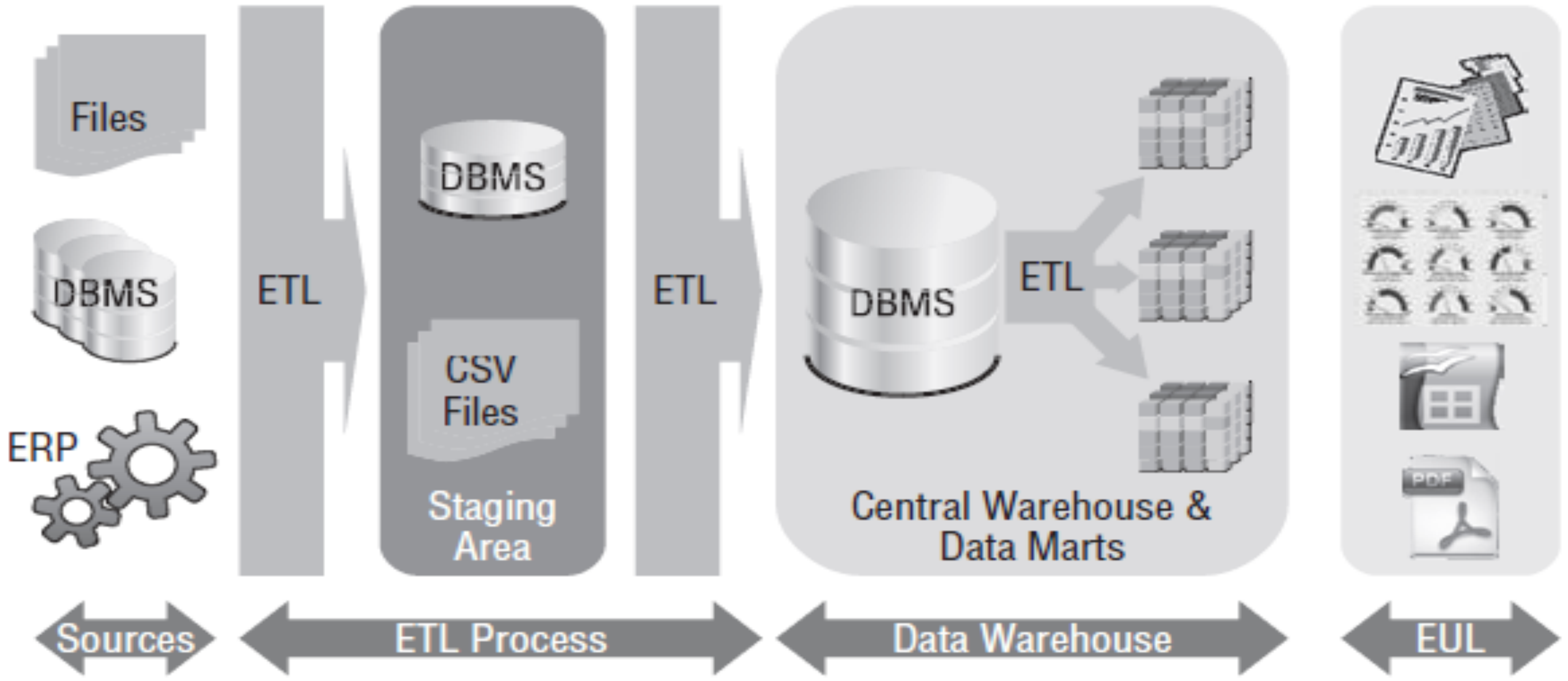
a.a 2017/2018

Prof.ssa G. Tortora

Three level architecture

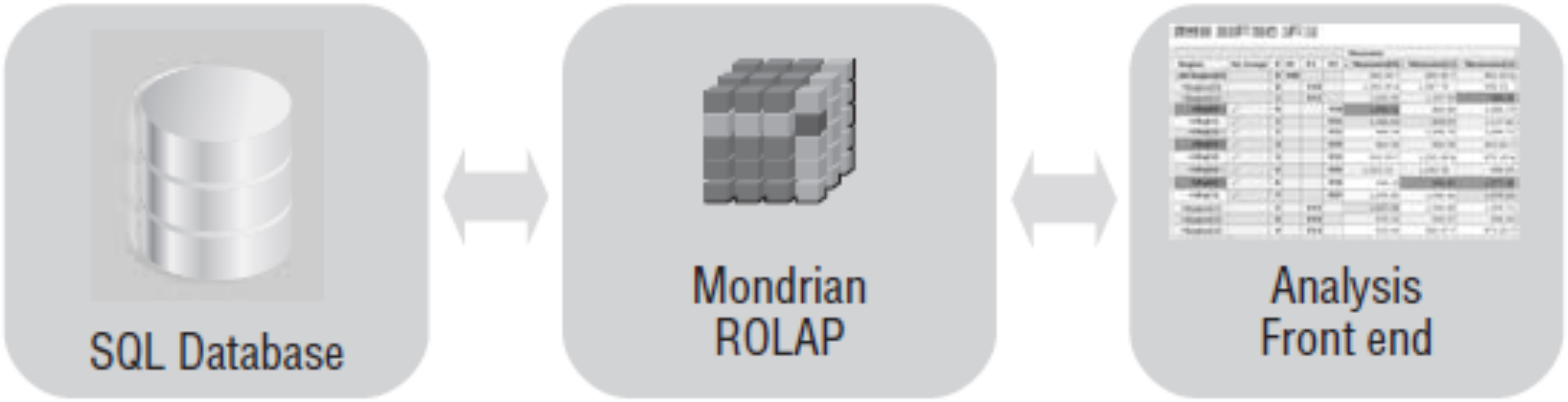


Generic data warehouse architecture

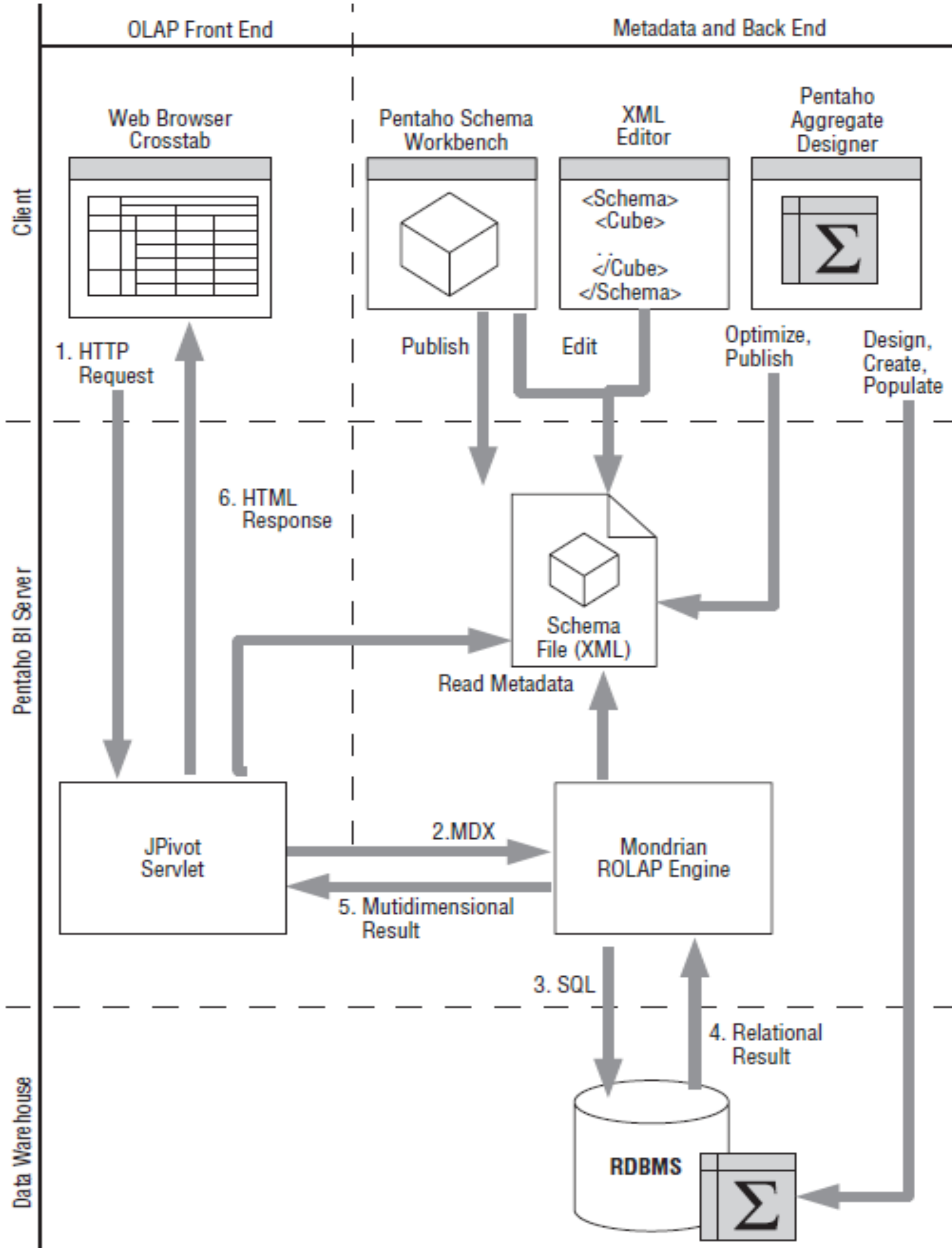


Data warehouse with Mondrian

- SQL Database: **MySQL**
- OLAP Engine: **Mondrian ROLAP**
- Analisys front end: **JPivot**



Pentaho OLAP components



Tools

- *JPivot analysis front end*:
  - JPivot is a Java-based analysis tool that serves as the actual user interface for working with OLAP cubes.
- *Mondrian ROLAP engine*:
  - The engine receives MDX (**M**ulti **D**imensional E**X**pressions) queries from front-end tools such as JPivot, and responds by sending a multidimensional result-set.
- *Schema Workbench*:
  - This is the visual tool for designing and testing Mondrian cube schemas. Mondrian uses these cube schemas to interpret MDX and translate it into SQL queries to retrieve the data from an RDBMS.
- *Data Integration*:
  - The desktop tool (Kettle) for building ETL jobs and transformations.

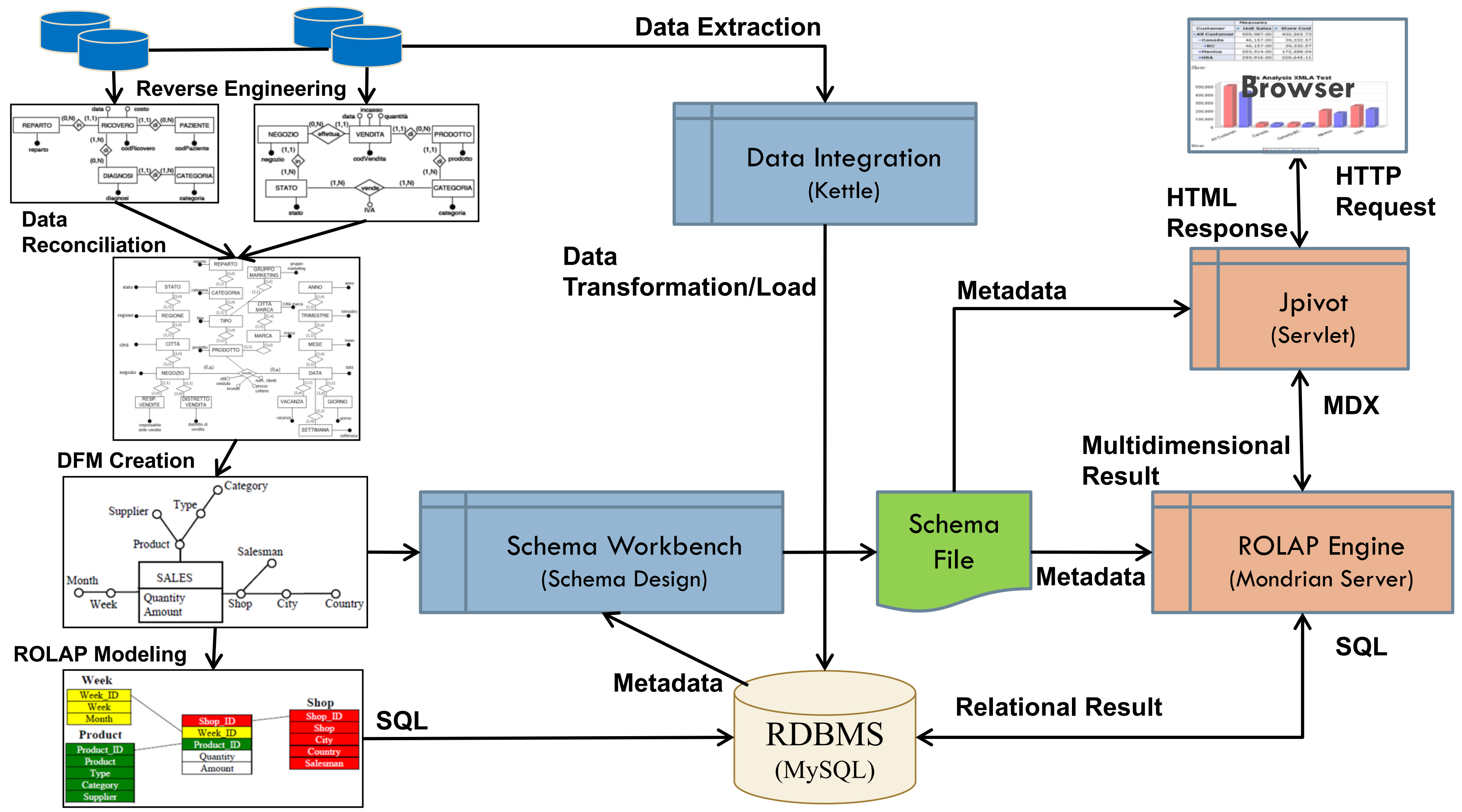
Schema

- A central structure is the *schema*.
  - The schema is essentially an XML document that describes one or more multidimensional cubes.
  - The cubes also describe the mapping of the cube’s dimensions and measures to tables and columns in a relational database.
  - To Mondrian, the schema is key in translating the MDX query to SQL queries.

Schema Design Tools

- The **multidimensional model**, consisting of dimensions, hierarchies, and measures, is created first and the relational model is mapped into the schema.
- Pentaho Schema Workbench offers a graphical user interface to create Mondrian schemas.
  - In addition, Pentaho Schema Workbench can publish schemas to the Pentaho Server, which then stores them in the solution repository.
  - Once stored in the solution repository, the schemas can be used by the server's Mondrian engine as a back end for OLAP services.

Data warehouse in practice *(with Mondrian)*



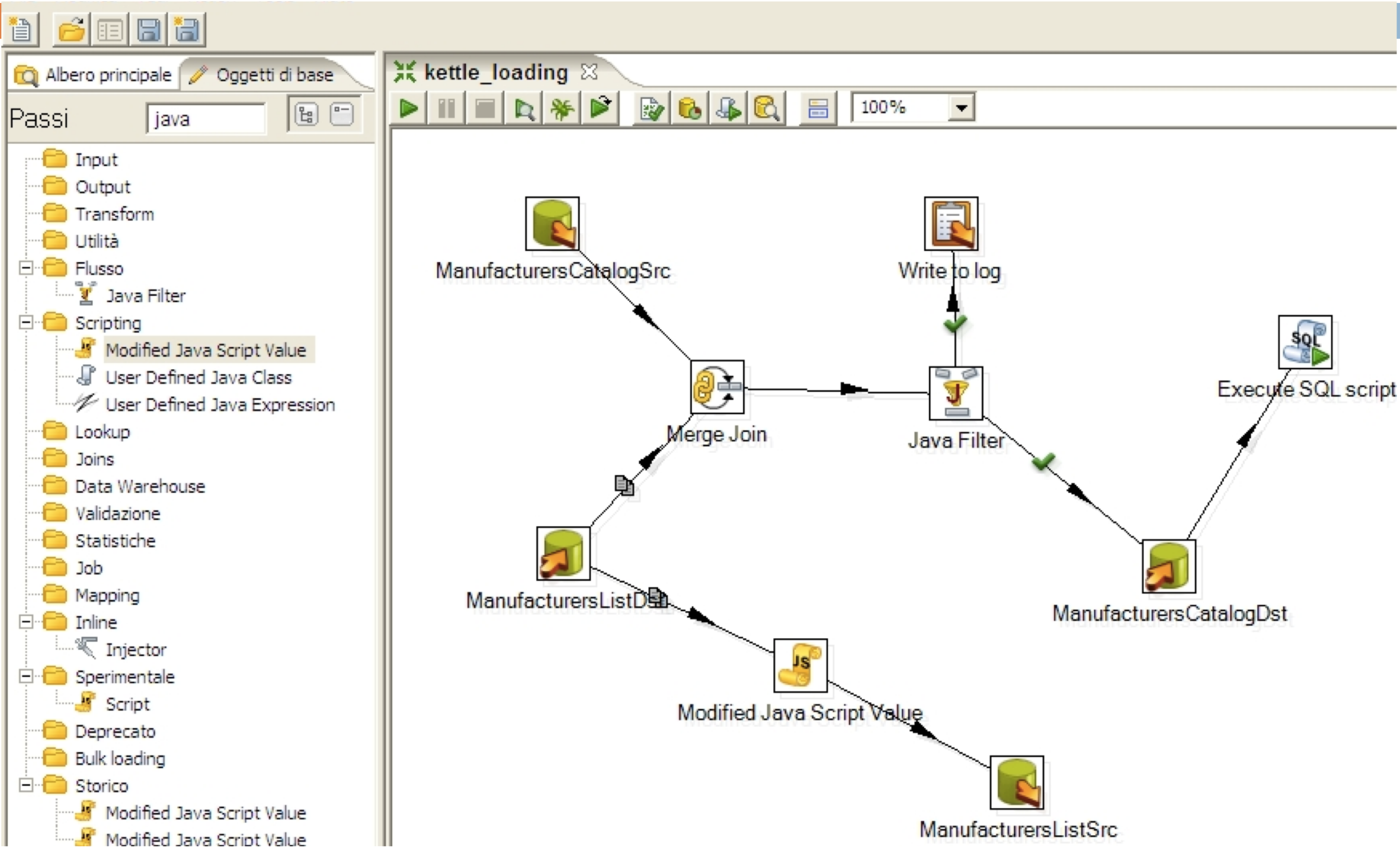
Introduction

- **P**entaho **D**ata **I**ntegration (PDI, also called *Kettle*) is the component of Pentaho responsible for the *Extract*, *Transform* and *Load* (ETL) processes.
- Though ETL tools are most frequently used in data warehouses environments, Kettle can also be used for other purposes:
  - Migrating data between applications or databases
  - Exporting data from databases to flat files
  - Loading data massively into databases
  - Data cleansing
  - Integrating applications
- Kettle is easy to use.
- Every process is created with a graphical tool where you specify what to do without writing code to indicate how to do it.

What is Spoon?

- Kettle is an acronym for *"Kettle E.T.T.L. Environment."* Kettle is designed to help you the Extraction, Transformation, Transportation and Loading of data.
- **Spoon** is a graphical user interface that allows you to design *transformations* and *jobs* that can be run with the Kettle tools — Pan and Kitchen.
- **Pan** is a data transformation engine that performs a multitude of functions such as reading, manipulating, and writing data to and from various data sources.
- **Kitchen** is a program that executes jobs designed by Spoon in XML or in a database repository.
  - Jobs are usually scheduled in batch mode to be run automatically at regular intervals.

Kettle UI


Albero principale
Oggetti di base
Passi
java
Input
Output
Transform
Utilità
Flusso
Java Filter
Scripting
Modified Java Script Value
User Defined Java Class
User Defined Java Expression
Lookup
Joins
Data Warehouse
Validazione
Statistiche
Job
Mapping
Inline
Injector
Sperimentale
Script
Deprecato
Bulk loading
Storico
Modified Java Script Value
Modified Java Script Value
kettle_loading
100%
ManufacturersCatalogSrc
Write to log
Execute SQL script
Merge Join
Java Filter
ManufacturersListDst
ManufacturersCatalogDst
Modified Java Script Value
ManufacturersListSrc

ETL by Example

- Kettle can be used as a standalone application, or it can be used as part of the larger Pentaho Suite.
- *As an ETL tool, it is the most popular open source tool available.*
- Kettle supports a vast array of input and output formats, including text files, data sheets, and commercial and free database engines.
- *Through a simple "Hello world" example, we will show how easy it is to work with Kettle and get you ready to make your own more complex transformations*.

Installing Kettle

- Follow the instructions below to install Spoon:

1. You can download Kettle (4.1 or higher) from http://kettle.pentaho.com/.
2. Install the Sun Microsystems Java Runtime Environment version 1.5 or higher.
3. Unzip the binary distribution zip-file in a directory of your choice.

Repository and files

- In Spoon, you build **Jobs** and **Transformations**.
- Kettle offers two methods to save them:
  - Database repository
  - Files
- If you choose the repository method, the repository has to be created the first time you execute Spoon.
- If you choose the files method, the Jobs are saved in files with the **kjb** extension, and the Transformations are in files with the **ktr** extension.
  - *We will work with the second method.*

Starting Spoon

- Start Spoon by executing **spoon.bat** on Windows, or **spoon.sh** on Unix-like operating systems.
- As soon as Spoon starts, a dialog window appears asking for the repository connection data…
- Go to the **Tools** menu and click **Options...**.
  - A window will come up that enables you to change various general and visual characteristics.
  - If you change something, it will be necessary to restart Spoon in order to see the changes applied.

Hello World Example

- Although this will be a simple example, it will introduce you to some of the fundamentals of Kettle:
  - Working with the Spoon tool
  - Transformations
  - Steps and Hops
  - Predefined variables
  - Previewing and Executing from Spoon
  - Executing Transformations from a terminal window with the Pan tool.

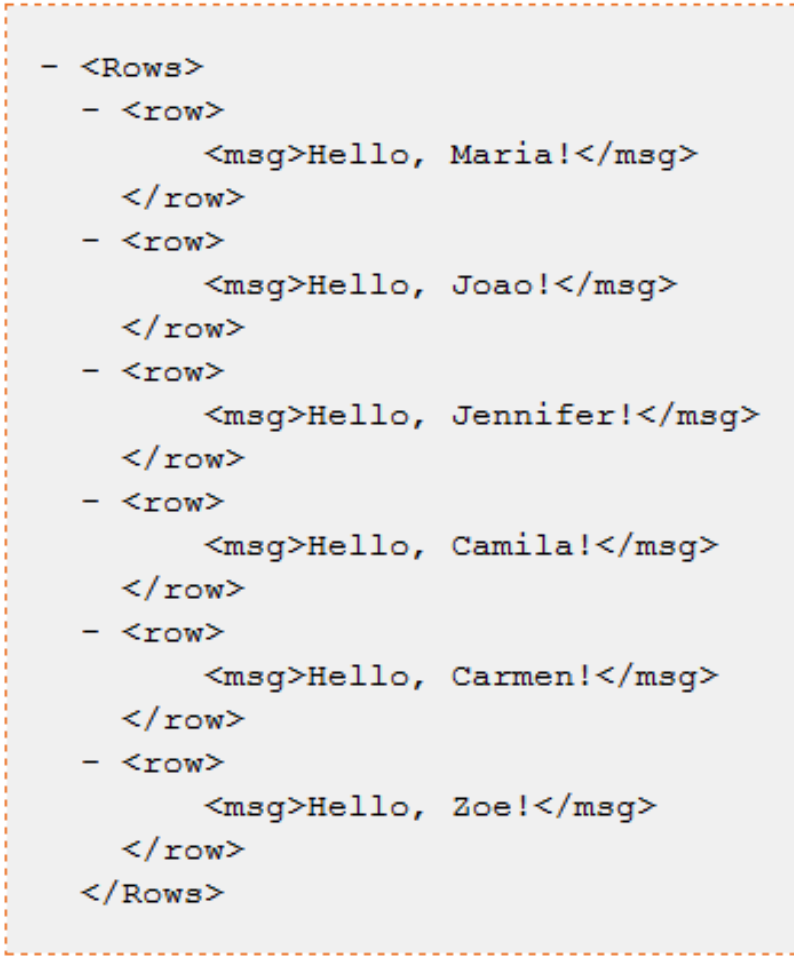
Overview

- Let's suppose that you have a CSV file containing a list of people, and want to create an XML file containing greetings for each of them.
- If this were the content of your CSV file:

```
last_name, name
Suarez,Maria
Guimaraes,Joao
Rush,Jennifer
Ortiz,Camila
Rodriguez,Carmen
da Silva,Zoe
```

Overview (2)

- This would be the output in your XML file:



```
- <Rows>
  - <row>
        <msg>Hello, Maria!</msg>
    </row>
  - <row>
        <msg>Hello, Joao!</msg>
    </row>
  - <row>
        <msg>Hello, Jennifer!</msg>
    </row>
  - <row>
        <msg>Hello, Camila!</msg>
    </row>
  - <row>
        <msg>Hello, Carmen!</msg>
    </row>
  - <row>
        <msg>Hello, Zoe!</msg>
    </row>
  </Rows>
```

Overview (3)

- The creation of the file with greetings from the flat file will be the goal for your first Transformation.
- A Transformation is made of **Steps** linked by **Hops**.
- These Steps and Hops form paths through which data flows:
  - Therefore it's said that a Transformation is *data-flow oriented*.

Preparing the environment

- Before starting a Transformation, create a **Tutorial** folder in the installation folder or some other convenient place.
- There you'll save all the files for this tutorial.
- Then create a CSV file like the one shown above, and save it in the Tutorial folder as **list.csv**.

Transformation walkthrough

- The proposed task will be accomplished in three subtasks:
  1. Creating the Transformation
  2. Constructing the skeleton of the Transformation using Steps and Hops
  3. Configuring the Steps in order to specify their behavior

Creating the Transformation

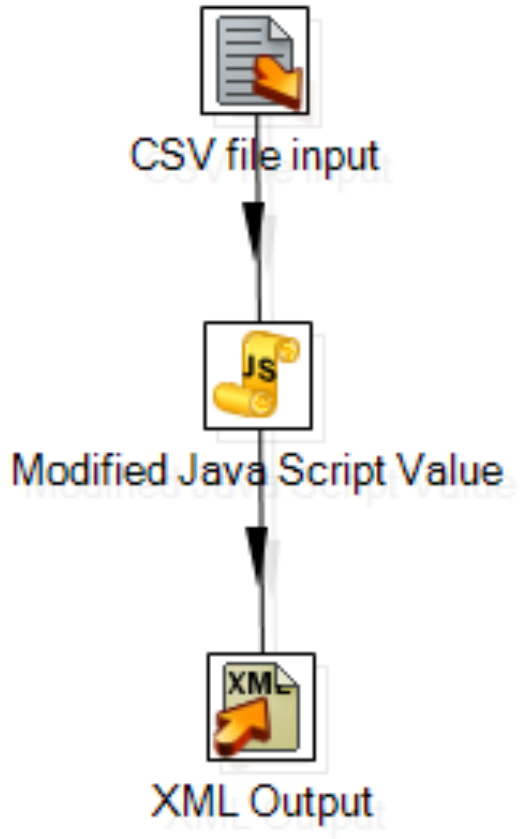
1. Click **New**, then select **Transformation**. Alternatively you can go to the **File** menu, then select **New**, then **Transformation**.
   - You can also just press **Ctrl-N**.
2. In the **View** navigator, click **Transformation 1**, then click **Settings**. Or right click the diagram and click **Transformation Settings**.
   - Or use the Ctrl+T shortcut.
3. A window appears where you can specify Transformation properties. In this case, just write a name and a description, then click **Save**.
4. Save the Transformation in the **Tutorial** folder with the name **hello**. This will create a **hello.ktr** file.

Constructing the skeleton of the Transformation using Steps and Hops

- A ***Step*** is the minimal unit inside a Transformation.
- A wide variety of Steps are available, grouped into categories like Input and Output, among others.
- Each Step is designed to accomplish a specific function, such as reading a parameter or normalizing a dataset.
- A ***Hop*** is a graphical representation of data flowing between two Steps, with an origin and a destination.
- The data that flows through that Hop constitutes the ***Output Data*** of the origin Step, and the ***Input Data*** of the destination Step.
- A Hop has only one origin and one destination, but more than one Hop could leave a Step.
  - When that happens, the Output Data can be copied or distributed to every destination.
- Likewise, more than one Hop can reach a Step.
  - In those instances, the Step has to have the ability to merge the Input from the different Steps in order to create the Output.

The Transformation

- A Transformation has to do the following:
  1. Read the CSV file
  2. Build the greetings
  3. Save the greetings in the XML file
- For each of these items you'll use a different Step, according to the next diagram:



The Transformation (2)

- Here's how to start the Transformation:
  1. To the left of the workspace is the **Steps Palette**. Select the **Input** category.
  2. Drag the CSV file onto the workspace on the right.
  3. Select the **Scripting** category.
  4. Drag the **Modified JavaScript Value** icon to the workspace.
  5. Select the **Output** category.
  6. Drag the **XML Output** icon to the workspace.

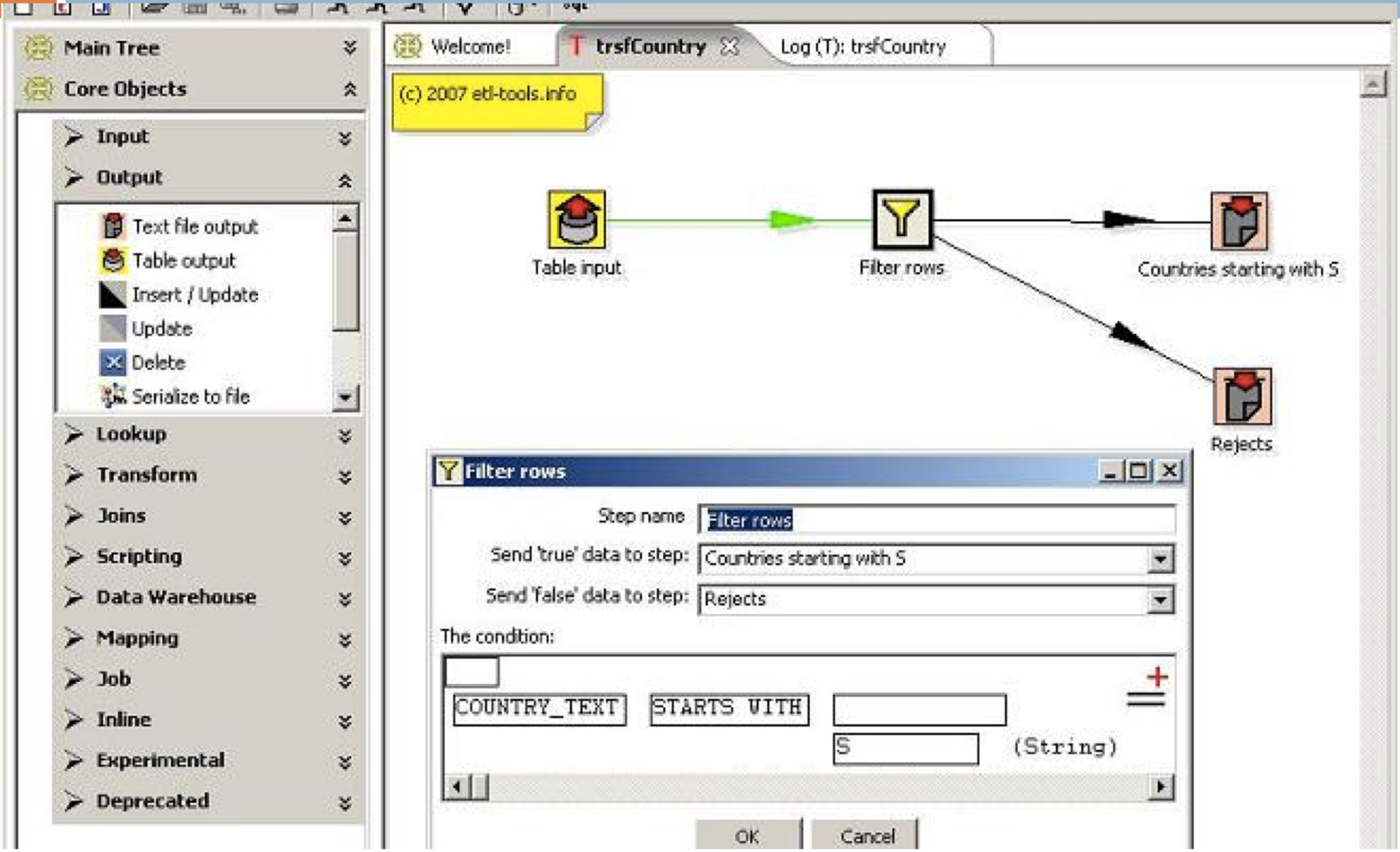
The Transformation (3)

- Now you will link the CSV file input with the Modified Java Script Value by creating a Hop:
  1. Select the first Step.
  2. Hold the **Shift** key and drag the icon onto the second Step.
  3. Link the Modified Java Script Value with the XML Output via this same process.

Specifying Step behavior

- Every Step has a configuration window.
- These windows vary according to the functionality of the Steps and the category to which they belong.
  - *Step Name* is always a representative name inside the Transformation - this doesn't change among Step configurations.
  - *Step Description* allows you to clarify the purpose of the Step.

The configuration window



Filter rows

Step name: Filter rows

Send 'true' data to step: Countries starting with S

Send 'false' data to step: Rejects

The condition:

COUNTRY_TEXT STARTS WITH S (String)

Configuring the CSV file input Step

1. Double-click on the CSV file input Step.
2. The configuration window belonging to this kind of Step will appear. Here you'll indicate the location, format and content of the input file.
3. Replace the default name with one that is more representative of this Step's function. In this case, type in **name list**.
4. In the **Filename** field, type the name and location of the input file.

Note

- It is possible to use variables as well as plain text in a field.
- A variable can be written manually as **${variable_name}** or selected from the variable window, which you can access by pressing **Ctrl-Spacebar**.
- This window shows both predefined and user-defined variables. Select:
  - *${Internal.Transformation.Filename.Directory}*
- Then type a slash and the name of the file you created:
  - *${Internal.Transformation.Filename.Directory}/list.csv*
- At runtime the variable will be replaced by its value, which will be the path where the Transformation was saved. The Transformation will search the file **list.csv** in that location.

Configuring the CSV file input Step (2)

5. Click **Get Fields** to add the list of column names of the input file to the grid. By default, the Step assumes that the file has headers (the **Header row present** checkbox is checked).
6. **Switch lazy conversion off.** When enables, lazy conversion avoids unnecessary data type conversions and can result in a significant performance improvements.
7. Click **Preview** to ensure that the file will be read as expected. A window showing data from the file will appear.
8. Click **OK** to finish defining the Step CSV file input.

Configuring the Modified JavaScript Value Step

1. Double-click on the **Modified JavaScript Value** Step.
2. The Step configuration window will appear, that allows you to write JavaScript code.
3. Name this Step **Greetings**.
4. The main area of the configuration window is for coding. To the left, there is a tree with a set of available functions that you can use in the code. Write the following code:

   ```
   var msg = 'Hello, ' + name + "!";
   ```

Configuring the Modified JavaScript Value Step (2)

5. At the bottom you can type any variable created in the code. In this case, you have created a variable named **msg**. Since you need to send this message to the output file, you have to write the variable name in the grid.
6. Click **OK** to finish configuring **Step Modified Script Value**.
7. Select the Step you just configured. In order to check that the new field will leave this Step, you will now see the Input and Output Fields.
8. Right-click the Step to bring up a context menu.
9. Select **Show Input Fields**. You'll see that the Input Fields are **last_name** and **name**, which come from the CSV file input Step.
10. Select **Show Output Fields**. You'll see that not only do you have the existing fields, but also the new **msg** field.

Note

- There are Steps that simply transform the input data. In this case, the input and output fields are usually the same.
- There are Steps, however, that add fields to the Output - **Calculator**, for example.
- There are other Steps that filter or combine data causing that the Output has less fields that the Input - **Group by**, for example.

Configuring the XML Output Step

1. Double-click the **XML Output Step**. The configuration window for this kind of Step will appear. Here you're going to set the name and location of the output file, and establish which of the fields you want to include. You may include all or some of the fields that reach the Step.
2. Name the Step **File with Greetings**.
3. In the **File** box write:

${Internal.Transformation.Filename.Directory}/Hello.xml

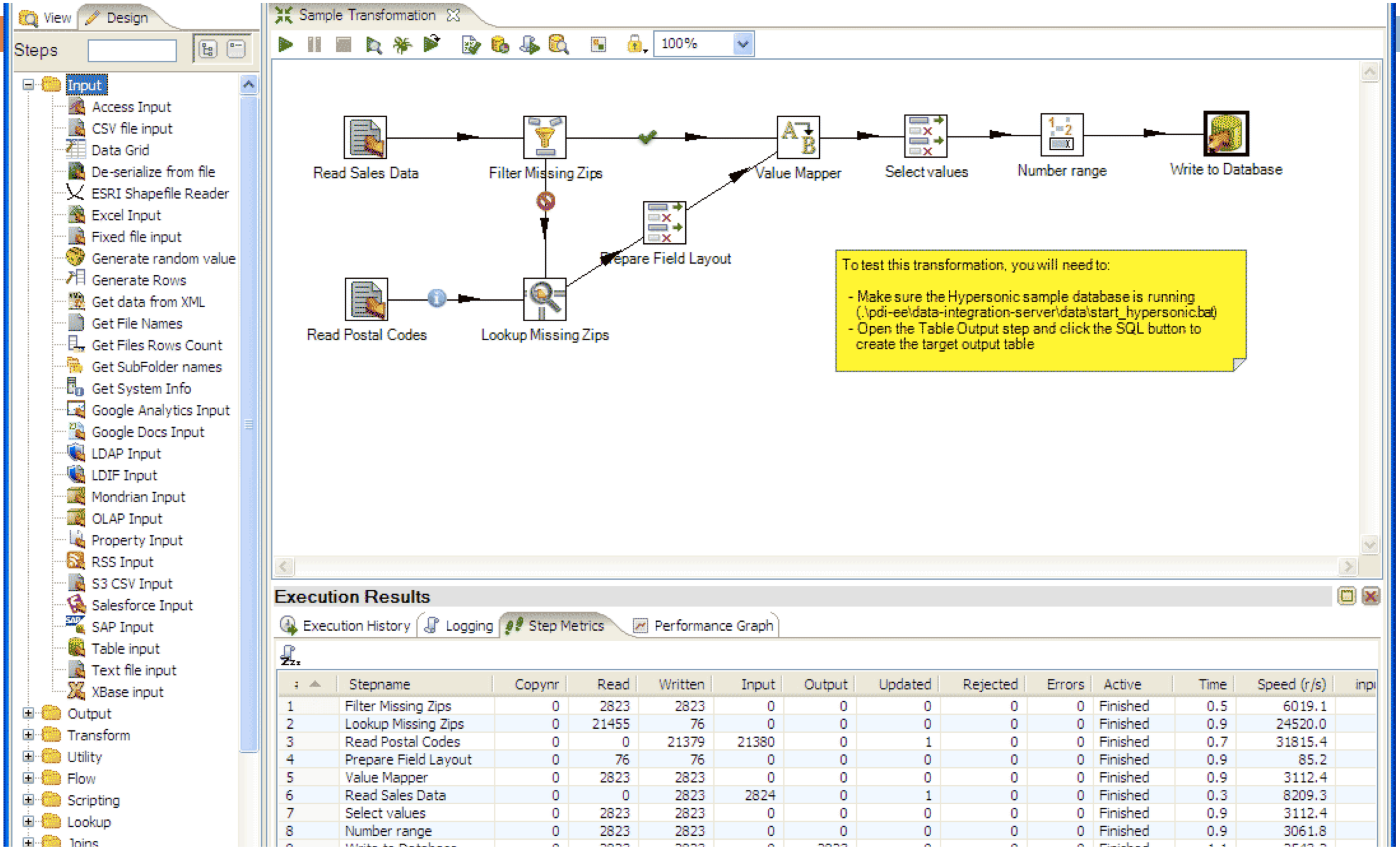
4. Click **Get Fields** to fill the grid with the three input fields, so delete **name** and **last_name**.

- Save the Transformation again.

How does it work?

- When you execute a Transformation, almost all Steps are executed simultaneously.
  - The Transformation executes asynchronously; the rows of data flow through the Steps at their own pace.
  - Each processed row flows to the next Step without waiting for the others. In real-world Transformations, forgetting this characteristic can be a significant source of unexpected results.
- At this point, Hello World is almost completely configured.
- A Transformation reads the input file, then creates messages for each row via the JavaScript code, and then the message is sent to the output file.
- This is a small example with very few rows of names, so it is difficult to notice the asynchronous execution in action.
- Keep in mind, however, that it's possible that at the same time a name is being written in the output file, another is leaving the first Step of the Transformation.

Executing a transformation



To test this transformation, you will need to:

- Make sure the Hypersonic sample database is running (.\pdi-ee\data-integration-server\data\start_hypersonic.bat)
- Open the Table Output step and click the SQL button to create the target output table

#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time	Speed (r/s)	inp
1	Filter Missing Zips	0	2823	2823	0	0	0	0	0	Finished	0.5	6019.1	
2	Lookup Missing Zips	0	21455	76	0	0	0	0	0	Finished	0.9	24520.0	
3	Read Postal Codes	0	0	21379	21380	0	1	0	0	Finished	0.7	31815.4	
4	Prepare Field Layout	0	76	76	0	0	0	0	0	Finished	0.9	85.2	
5	Value Mapper	0	2823	2823	0	0	0	0	0	Finished	0.9	3112.4	
6	Read Sales Data	0	0	2823	2824	0	1	0	0	Finished	0.3	8209.3	
7	Select values	0	2823	2823	0	0	0	0	0	Finished	0.9	3112.4	
8	Number range	0	2823	2823	0	0	0	0	0	Finished	0.9	3061.8	
9	Write to Database	0	[illegible]	[illegible]	0	[illegible]	0	0	0	Finished	[illegible]	[illegible]	

Verify, preview and execute

- Before executing the Transformation, check that everything is properly configured by clicking **Verify**.
- Spoon will verify that the Transformation is syntactically correct, and look for unreachable Steps and nonexistent connections.
  - If everything is in order (it should be if you followed the instructions), you are ready to preview the output.

Preview and Execute

1. Select the JavaScript Step and then click **Preview** button.
2. As you can see, Spoon suggests that you preview the selected Step. Click **QuickLaunch**. After that, you will see a window with a sample of the output of the JavaScript Step.
    - If the output is what you expected, you're ready to execute the Transformation.
3. Click **Run**.
4. Spoon will show a window where you can set, among other information, the parameters for the execution and the logging level.
5. Click **Launch**. A new window tab will appear in the Job window. This is the log tab, which contains a log of the current execution.

Step metrics

- In the step metric section the executed operations for each Step of the Transformation are provided.
- In particular, pay attention to these:
  - **Read**: the number of rows coming from previous Steps.
  - **Written**: the number of rows leaving from this Step toward the next.
  - **Input**: the number of rows read from a file or table.
  - **Output**: the number of rows written to a file or table.
  - **Errors**: errors (in **red**) in the execution.

Log tab

- In the log tab you will see the execution step by step.
  - The detail will depend on the log level established.
  - If you pay attention to this detail, you will see the asynchronicity of the execution.
  - The last line of the text will be:

    ```
    Spoon - The transformation has finished!!
    ```

- If there weren't error messages in the text, open the newly generated **Hello.xml** file and check its content.

Pan

- Pan allows you to execute Transformations from a terminal window.
  - The script is **pan.bat** on Windows, or **pan.sh** on other platforms, and it's located in the installation folder.
  - If you run the script without any options, you'll see a description pan with a list of available options.
- To execute your Transformation, try the simplest command:

```
pan    /file <Jobs_path>/Hello.ktr   /norep
```

  - **/norep** is a command to ask Spoon not to connect to the repository.
  - **/file** precedes the name of the file that contains the Transformation.
  - **<Jobs_path>** is the full path to the Tutorial folder.

Refining Hello World

- Now that the Transformation has been created and executed, the next task is enhancing it.
- ***Exercise:*** *execute the Transformation you created, setting as the name of the input file, a file that doesn't exist. See what happens!*

Hello World Refined Example

- This example will introduce you to some of the fundamentals of Kettle:
  - Jobs
  - Job Entries and Hops
  - Input parameters
  - Setting variables
  - Conditions and branches
  - Executing Jobs from a terminal window with the Kitchen tool.

Overview

- These are the improvements that you'll make to your existing Transformation:
  - You won't look for the input file in the same folder, but in a new one, a folder independent to that where the Transformations are saved.
    - The name of the input file won't be fixed; the Transformation will receive it as a parameter.
  - You will validate the existence of the input file.
    - The name the output file will be dependent of the name of the input file.

The improvements

- Here's what happens:
  - Get the parameter
  - Check if the parameter is null; if it is, abort
  - Check if the file exists; if not, abort
  - Create the output file with greetings

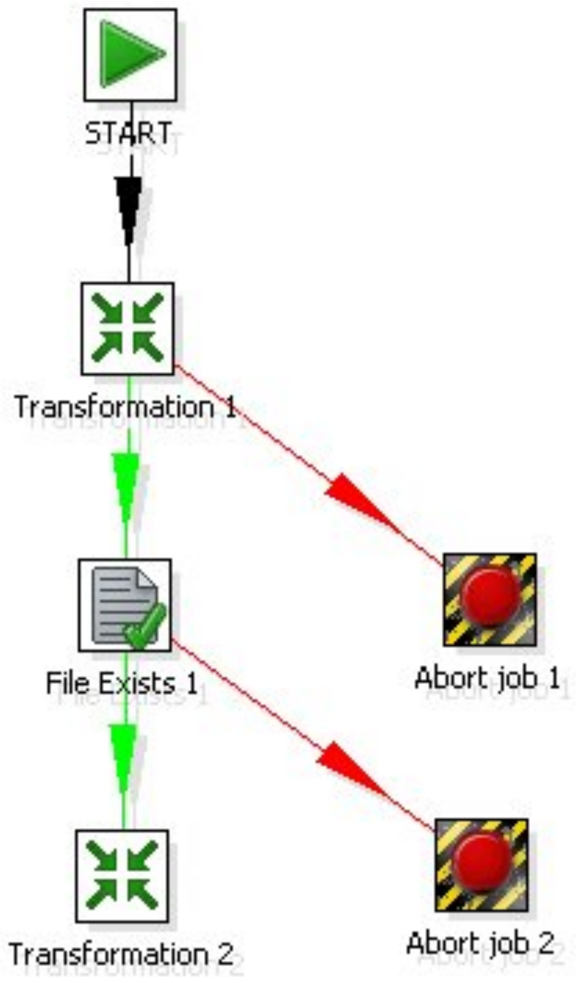
Job

- This will be accomplished via a *Job*, which is a component made by **Job Entries** linked by **Hops**.
  - These Entries and Hops are arranged according the expected order of execution. Therefore it is said that a Job is *flow-control oriented*.
- A **Job Entry** is a unit of execution inside a Job.
  - Each Job Entry is designed to accomplish a specific function, ranging from verifying the existence of a table to sending an email.
- From a Job it is possible to execute a Transformation or another Job, that is, Jobs and Transformations are also Job Entries.
- A **Hop** is a graphical representation that identifies the sequence of execution between two Job Entries.
  - Even when a Hop has only one origin and one destination, a particular Job Entry can be reached by more than a Hop, and more than a Hop can leave any particular Job Entry.

The process

- This is the process:
  1. Getting the parameter will be resolved by a new Transformation.
  2. The parameter will be verified through the result of the new Transformation, qualified by the conditional execution of the next Steps.
  3. The file's existence will be verified by a Job Entry.
  4. Executing the main task of the Job will be made by a variation of the Transformation you made in the first Hello World example.

Graphically



Preparing the Environment

- The input and output files will be in a new folder called **Files**.
  - Copy the **list.csv** file to this new directory.
- Create a variable containing this information. To do this, edit the **kettle.properties** configuration file.
- Put this line at the end of the file, changing the path to the one specific to the Files directory you just created:

```
FILES=<File_Path>/Files
```

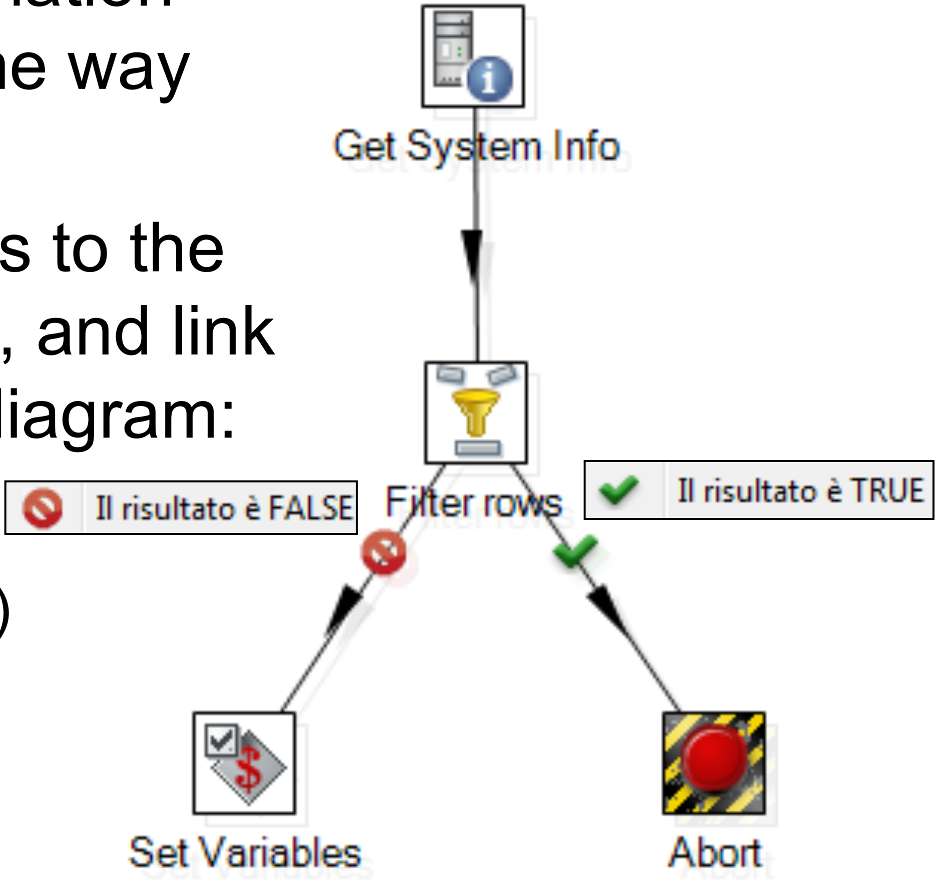
- Restart Spoon.

Todo

- *Now you are ready to start.*
- This process involves three stages:
  1. Create the Transformation
  2. Modify the Transformation
  3. Build the Job

Creating the Transformation

1. Create a new Transformation **get_file_name** the same way you did before.
2. Drag the following Steps to the workspace, name them, and link them according to the diagram:



- Get System Info (Input category)
- Filter Rows (Flow category)
- Abort (Flow category)
- Set Variable (Job category)

Configuring the Get System Info

- This Step captures information from sources outside the Transformation, like the system date or parameters entered in the command line.
  - We will use the Step to get the first and only parameter.
  - The configuration window of this Step has a grid. In this grid, each row you fill will become a new column containing system data.
- Double-click the Step.
- In the first cell, below the Name column, write **my_file.**
- When you click the cell below Type, a window will show up with the available options.
  - Select **command line argument 1**.
- Click **OK**.

Configuring the Filter Rows

- This Step divides the output in two, based upon a condition. Those rows for which the condition evaluates to true follow one path in the diagram, the others follow another.
- Double-click the Step.
- *Write the condition:* In **Field** select **my_file** and replace the **=** with **IS NULL**.
- In the drop-down list next to **Send 'true' data to Step**, select **Abort**.
- In the drop-down list next to **Send 'false' data to Step**, select **Set Variable**.
- Click **OK**.
- Now a NULL parameter will reach the Abort Step, and a NOT NULL parameter will reach the Set Variable Step.

Configuring the Abort

- You don't have anything to configure in this Step. If a row of data reaches this Step, the Transformation aborts, then fails, and you will use that result in the main Job.

Configuring the "Set Variable"

- This Step allows you to create variables and put the content of some of the input fields into them.
  - The configuration window of the Step has a grid.
  - Each row in this grid is meant to hold a new variable.
- Now you'll create a new variable to use later:
  1. Double-click the Step.
  2. Click **Get Fields**. The only existing field will appear: **my_file**. The default variable name is the name of the selected field in upper case: **MY_FILE**. Leave the default intact.
  3. Click OK.

Execution

- To test the Transformation, click **Run**.
- Within the run dialog, you will find a grid titled "Arguments" on the bottom left.
  - Delete whatever arguments are already inside, and instead type **list** as the first argument value. This will be transferred to the transformation as the command line argument.
- Click **Launch**.
- In the *Logging* pane, you'll see a message like this:
  - Set Variables.0 - Set variable MY_FILE to value [list]
- Click **Run** again, and clear the value of the first argument. This time, when you hit **Launch** you'll see this:
  - Abort.0 - Row nr 1 causing abort : []
  - Abort.0 - Aborting after having seen 1 rows.

Modifying the Transformation

- Now it's time to modify the **Hello** transformation in order to match the names of the files to their corresponding parameters.
- If the command line argument to the job would be **bd2**, this transformation should read the file **bd2.csv** and create the file **bd2_with_greetings.xml**.
  - It would also be helpful to add a filter to discard the empty rows in the input file.
- Open the Transformation **Hello.ktr**.
- Open the **CSV File Input Step** configuration window.
- Delete the content of the **Filename** text box, and press **Ctrl-Spacebar** to see the list of existing variables. You should see the **FILES** variable you added to kettle.properties. The text becomes:

${FILES}/${MY_FILE}.csv

Modifying the Transformation (2)

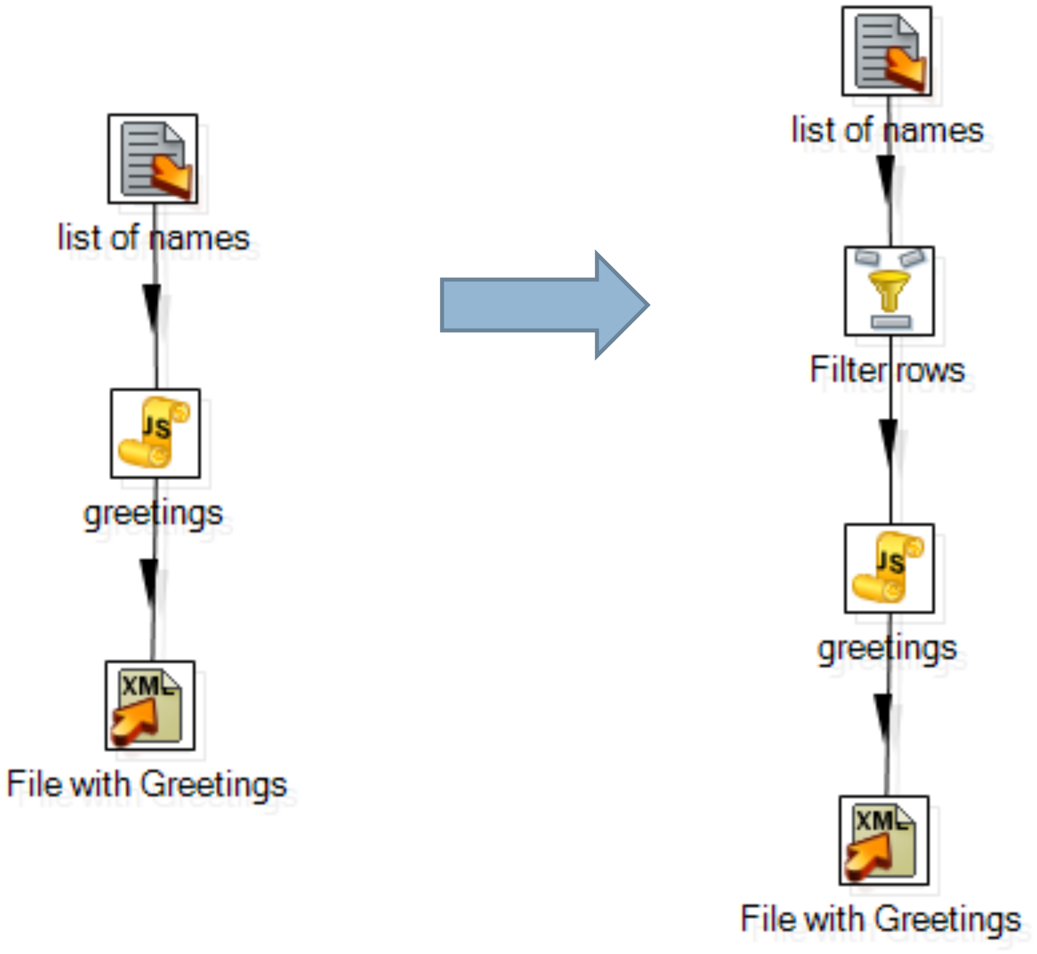
- Open the **XML Output Step** configuration window.
- Replace the content of the **Filename** text box with this:

  ${FILES}/${MY_FILE}_with_greetings
- Click **Show Filename(s)** to view the projected XML filename.

Modifying the Transformation (3)

- Drag a **Filter Rows** step into the transformation.
- Drag the **Filter Rows** step onto the Hop that leaving **CSV Input** and reaching **Modified Javascript Script Value**.
  - When the Hop line becomes emphasized (thicker), release the mouse button.
  - You have now linked the new step to the sequence of existent steps.
- Select **name** for the Field, and **IS NOT NULL** for the comparator.
- Leave **Send 'true' data to Step** and **Send 'false' data to Step** blank.
  - This makes it so only the rows that fulfill the condition (rows with non-null names) follow to the next Step. This is similar to an earlier Step.
- Click **OK**.
- Click **Save As** and name this Transformation **Hello_with_parameters**.

Graphically



Executing the Transformation

- To test the changes you made, you need to make sure that the variable **MY_FILE** exists and has a value.
  - Because this Transformation is independent of the Transformation that creates the variable, in order to execute it, you'll have to create the variable manually.
- In the **Edit** menu, click **Set Environment Variables**.
  - A list of variables will appear.
  - At the bottom of the list, type in **MY_FILE** as the variable name; as the content, type the name of the file (i.e., *list*) without its extension.
- Click **OK**.
- Click **Run**.
- In the list of variables, you'll see the one you just created. Click **Launch** to execute the Transformation.
- Lastly, verify the existence and content of the output file.

Building the main job

- Create the Job:
  1. Click **New**, then **Job**.
  2. The Job workspace, where you can drop Job Entries and Hops, will come up.
  3. Click **Job**, then **Settings**.
  4. A window in which you can specify some Job properties will come up.
     - Type in a name and a description.
  5. Click **Save**. Save the Job in the Tutorial folder, under the name **Hello**.

Building the main job (2)

- Build the skeleton of the Job with Job Entries and Hops:
  1. Drag the following steps into the workspace: one **General->Start** step, two **General->Transformation** steps, and one **File Exists** step.
  2. Link them in the following order: Start, Transformation, File Exists, Transformation.
  3. Drag two **General->Abort** steps to the workspace. Link one of them to the first **Transformation** step and the other to the **File Exists** step.
     - The newly created hops will turn red.

Configure the Steps

- Double click the first Transformation step. The configuration window will come up.
- In the **Transformation filename** field, type the following:

  ${Internal.Job.Filename.Directory}/get_file_name.ktr

- This will work if transformations and jobs reside in the same folder.
- Click **OK**.

Configure the second Transformation

- Double-click the entry. The configuration window will come up.
- Type the name of the other Transformation in the **Transformation Filename** field:

  ${Internal.Job.Filename.Directory}/Hello_with_parameter.ktr

- Click **OK**.

Configure the File Exists

- Double-click the entry to bring up the configuration window.
- Put the complete path of the file whose existence you want to verify in the **Filename** field.
- The name is the same that you wrote in the modified Transformation Hello:

  ${FILES}/${MY_FILE}.csv

Configure the Abort steps

- Configure the first Abort step:
  - In the Message textbox write:
    - **The file name argument is missing.**

- Configure the second Abort step:
  - In the Message textbox write this text:
    - **The file ${FILES}/${MY_FILE}.csv does not exist.**

- **Note:** In runtime, the tool will replace the variable names by its values. If you place your mouse pointer over the Message textbox, Spoon will display a tooltip showing projected output.

Configuring the Hops

- A Job Entry can be executed unconditionally (it's executed always), when the previous Job Entry was successful, and when the previous Job Entry failed.
- This execution is represented by different colors in the Hops:
  - a **black** Hop indicates that the following Job Entry is always executed;
  - a **green** Hop indicates that the following Job Entry is executed only if the previous Job Entry was successful;
  - a **red** Hop indicates that the following Job Entry is executed only if the previous Job Entry failed.

Configuring the Hops (2)

- The Steps will execute as you need:
  - The first Transformation entry will be always executed.
  - If the Transformation that gets the parameter doesn't find a parameter, (that is, the Transformation failed), the control goes through the red Hop towards the Abort Job entry.
  - If the Transformation is successful, the control goes through the green Hop towards the File Exists entry.
  - If the file doesn't exist the control goes through the red Hop, towards the second Abort Job entry.
  - If the verification is successful, the control goes through the green Hop towards the main Transformation entry.

Configuring the Hops (3)

- If you wanted to change the condition for the execution of a Job Entry, the steps to follow would be:
  - Select the Hop that reached this Job Entry.
  - Right click to bring up a context menu.
  - Click **Evaluation**, then one of the three available conditions.

How it works

- When you execute a Job, the execution is tied to the order of the Job Entries, the direction of the Hops, and the condition under which an entry is or not executed. The execution follows a sequence. The execution of a Job Entry cannot begin until the execution of the Job Entries that precede it has finished.
- In real-world situations, a Job can be a solution to solve problems related to a sequence of tasks in the Transformations. If you need a part of a Transformation to finish before another part begins, a solution could be to divide the Transformation into two independent Transformations, and execute them from a Job, one after the other.

Executing the Job

- To execute a Job, you first must supply a parameter. Because the only place where the parameter is used is in the **get_file_name** Transformation (after that you only use the variable where the parameter is saved) write the parameter as follows:
  1. Double-click the **get_file_name** Transformation Step.
  2. The ensuing window has a grid named **Arguments**. In the first row type **list**.
  3. Click **OK**.
  4. Click the **Run** button, or from the title menu select **Job->Run**.
  5. A window will appear with general information related with the execution of the Job.
  6. Click **Launch**.
  7. The execution results pane on the bottom should display the execution results.

Executing the Job (2)

- Alternatively, to test the Job directly:
  1. Click the **Run** button, or from the title menu select **Job->Run**.
  2. Within the run dialog, you will find a grid titled "Arguments" on the bottom left.
     - Type **list** as the first argument value. This will be transferred to the transformation as the command line argument.
  3. A window will appear with general information related with the execution of the Job.
  4. Click **Launch**.

Executing the Job (3)

- The new file has been created when you see this at the end of the log text:

  Spoon - Job has ended.

- If the input file was **list.csv**, then the output file should be **list_with_greetings.xml** and should be in the same folder. Find it and check its content.
- Now change the name of the parameter by replacing it with a nonexistent file name or deleting the file name and execute the Job again.

Kitchen

- Kitchen is the tool used to execute Jobs from a terminal window. The script is **kitchen.bat** on Windows, and **kitchen.sh** on other platforms, and you'll find it in the installation folder.
- If you execute it, you'll see a description of the command with a list of the available options.
- To execute the Job, try the simplest command:

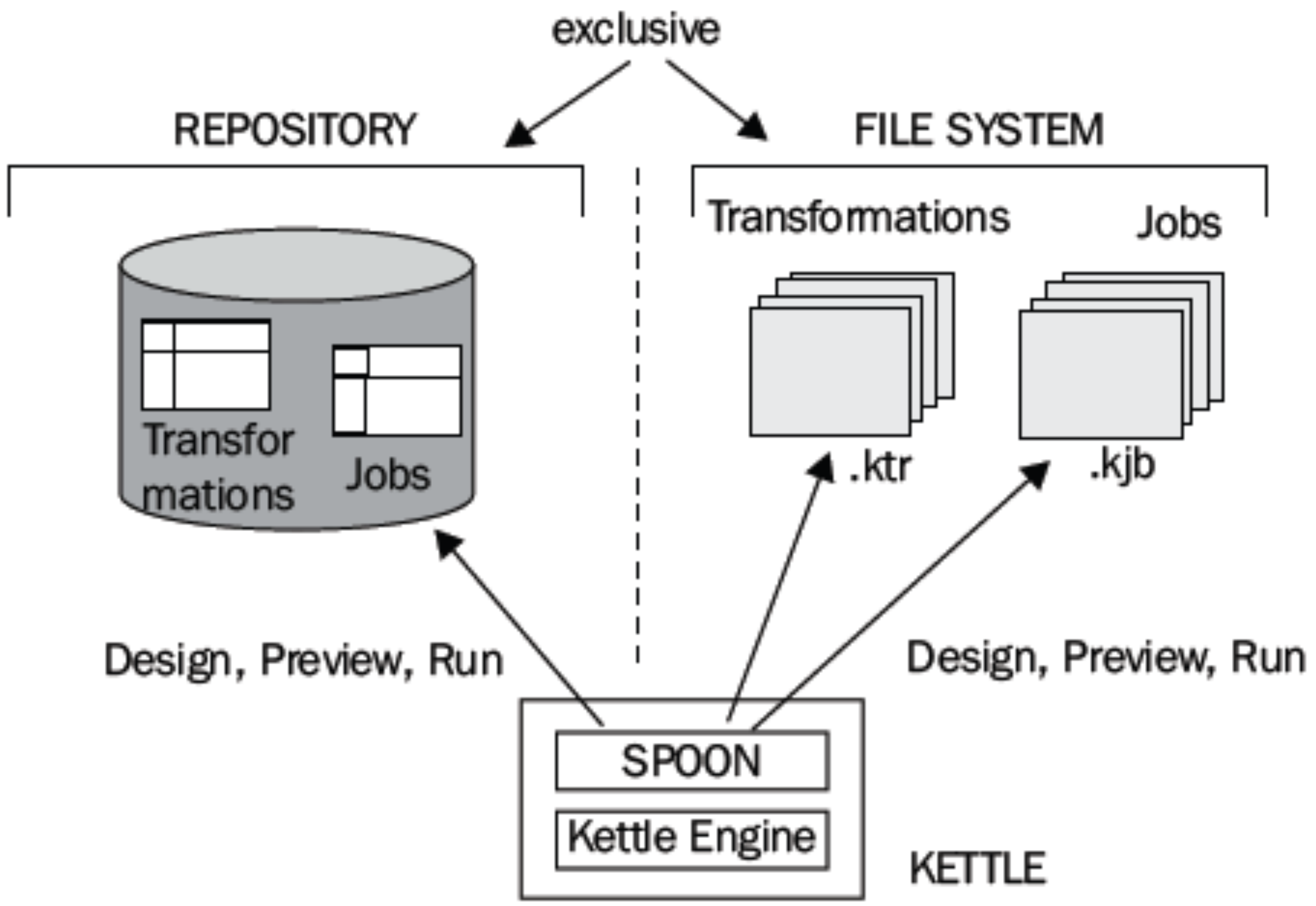
```
kitchen /file <Jobs_path>/Hello.kjb <par> /norep
```

- **<par>** is the parameter that the Job is waiting for.
  - Remember that the expected parameter is the name of the input file, without the csv.

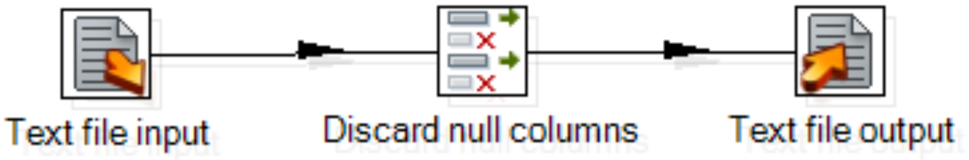
Storing transformations and jobs in a repository

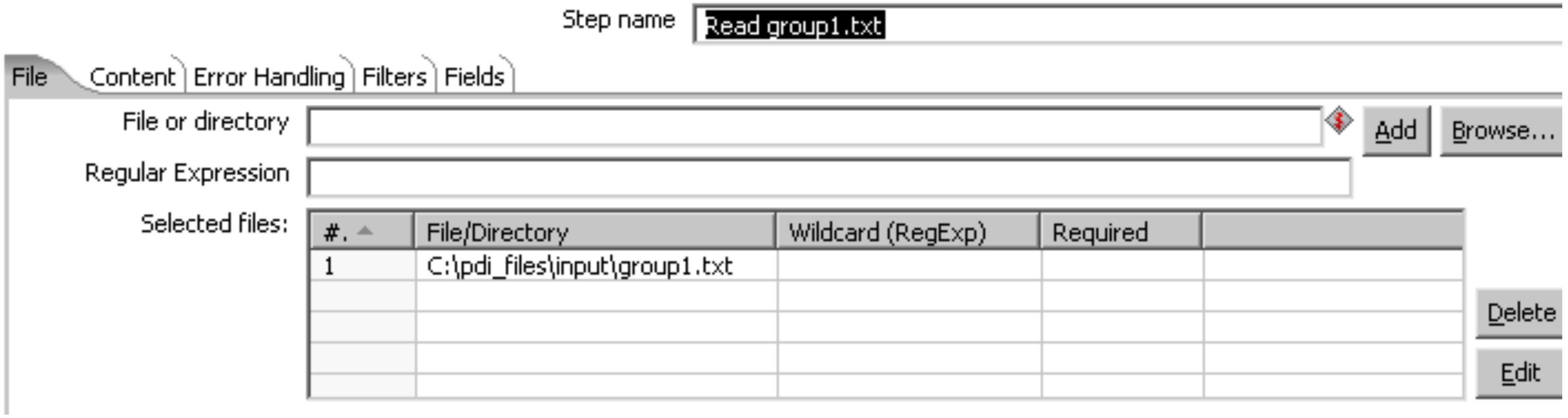
- The first time you launched Spoon, you chose **No Repository**.
- PDI offers two methods:
  - **Repository**: When you use the repository method you save jobs and transformations in a repository.
    - A repository is a relational database specially designed for this purpose.
  - **Files**: The files method consists of saving jobs and transformations as regular XML files in the file-system, with extension **kjb** and **ktr** respectively.

Repository/Files



Reading a formatted file





Step name: Read group1.txt

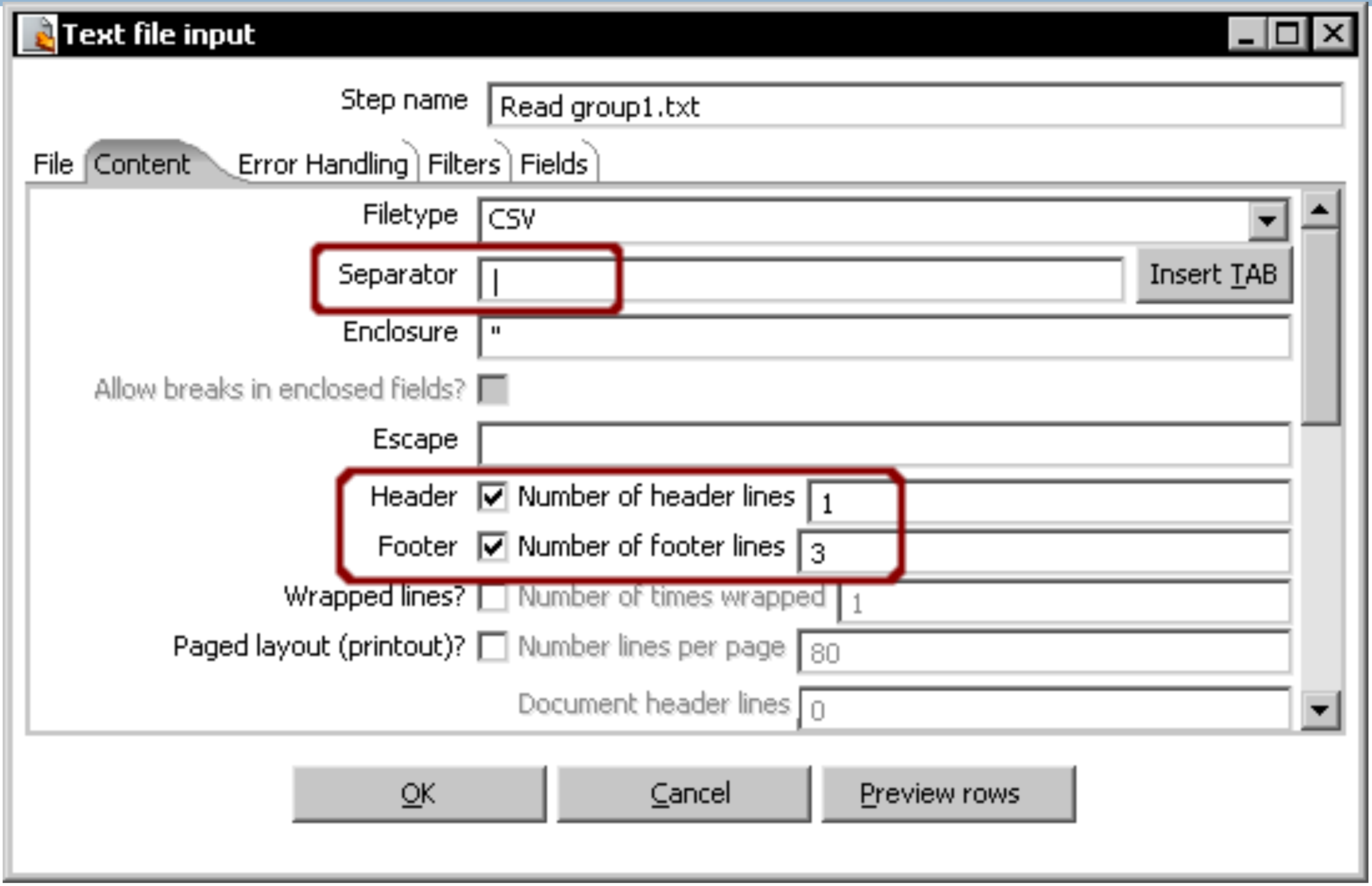
File or directory:

Regular Expression:

Selected files:

#.	File/Directory	Wildcard (RegExp)	Required
1	C:\pdi_files\input\group1.txt		

Text file input (content tab)



Text file input

Step name: Read group1.txt

File | Content | Error Handling | Filters | Fields

Filetype: CSV

Separator: |

Enclosure: "

Allow breaks in enclosed fields? ☐

Escape:

Header ☑ Number of header lines: 1

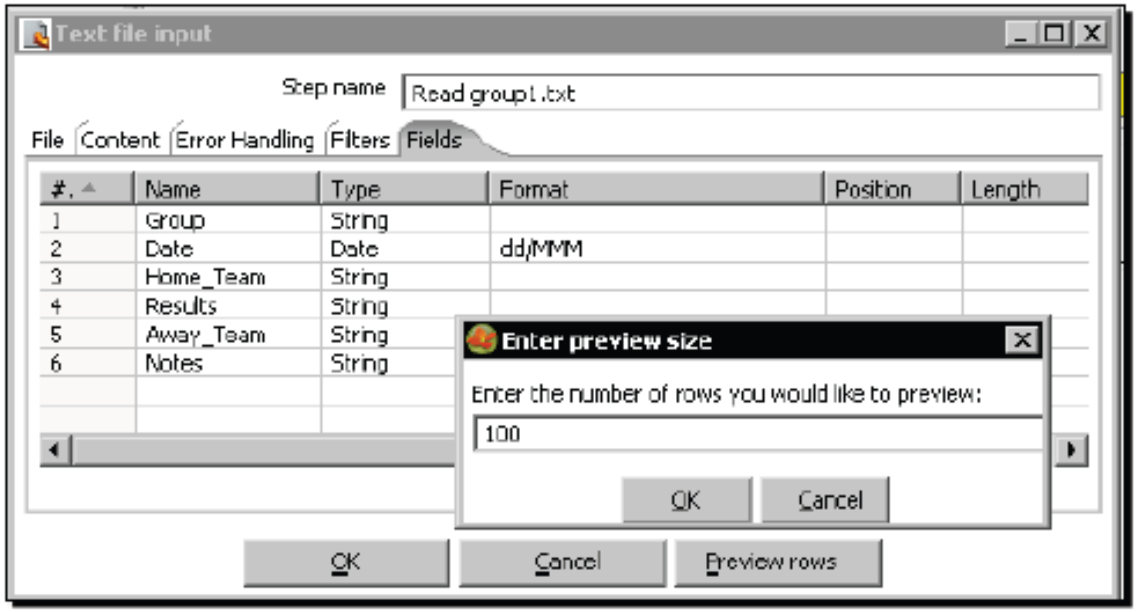
Footer ☑ Number of footer lines: 3

Wrapped lines? ☐ Number of times wrapped: 1

Paged layout (printout)? ☐ Number lines per page: 80

Document header lines: 0

Text file input (field tab)



Text file input

Step name: Read group1.txt

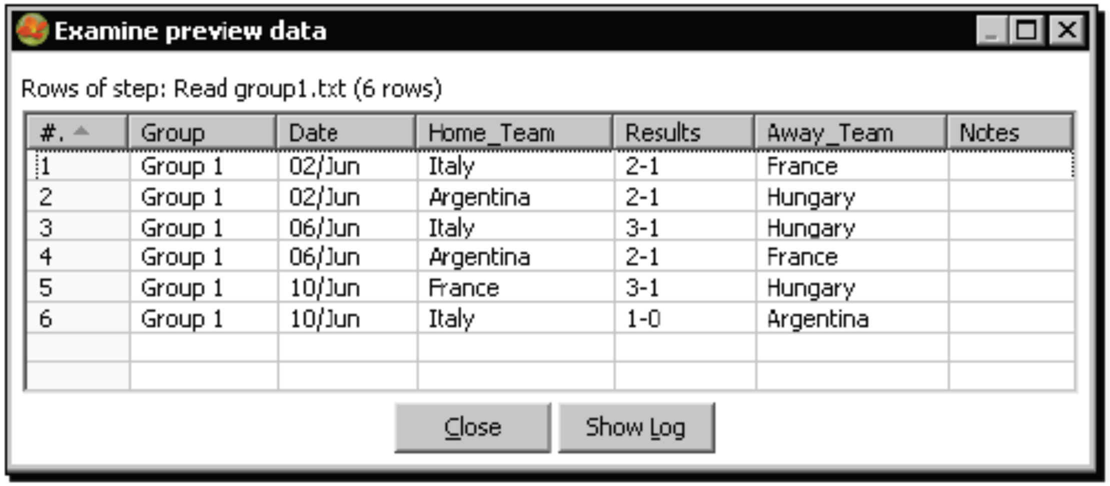
File | Content | Error Handling | Filters | Fields

#.	Name	Type	Format	Position	Length
1	Group	String			
2	Date	Date	dd/MMM		
3	Home_Team	String			
4	Results	String			
5	Away_Team	String			
6	Notes	String			

Enter preview size

Enter the number of rows you would like to preview:

100

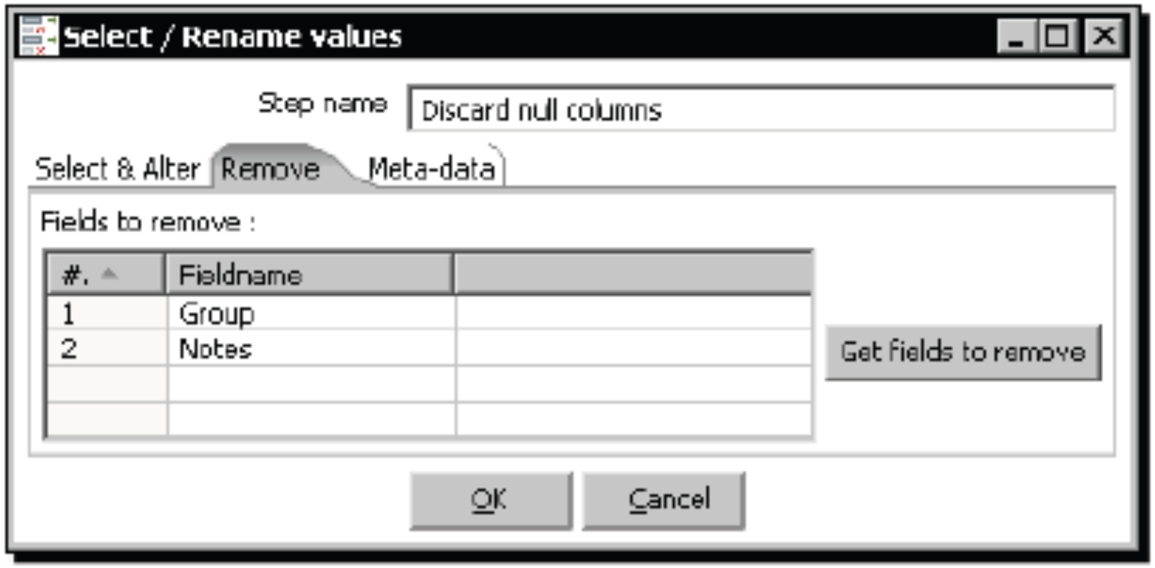


Examine preview data

Rows of step: Read group1.txt (6 rows)

#.	Group	Date	Home_Team	Results	Away_Team	Notes
1	Group 1	02/Jun	Italy	2-1	France	
2	Group 1	02/Jun	Argentina	2-1	Hungary	
3	Group 1	06/Jun	Italy	3-1	Hungary	
4	Group 1	06/Jun	Argentina	2-1	France	
5	Group 1	10/Jun	France	3-1	Hungary	
6	Group 1	10/Jun	Italy	1-0	Argentina	

Remove columns

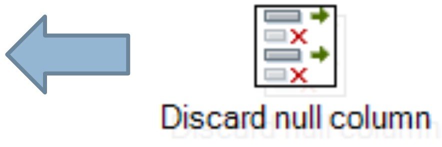

Select / Rename values

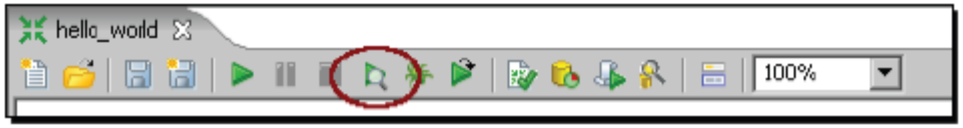
Step name: Discard null columns

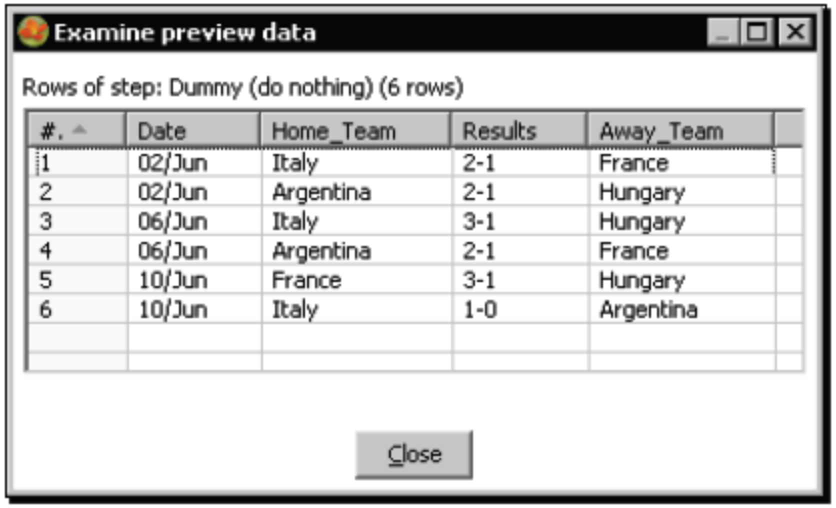
Select & Alter | Remove | Meta-data

Fields to remove :

#.	Fieldname	
1	Group	
2	Notes	



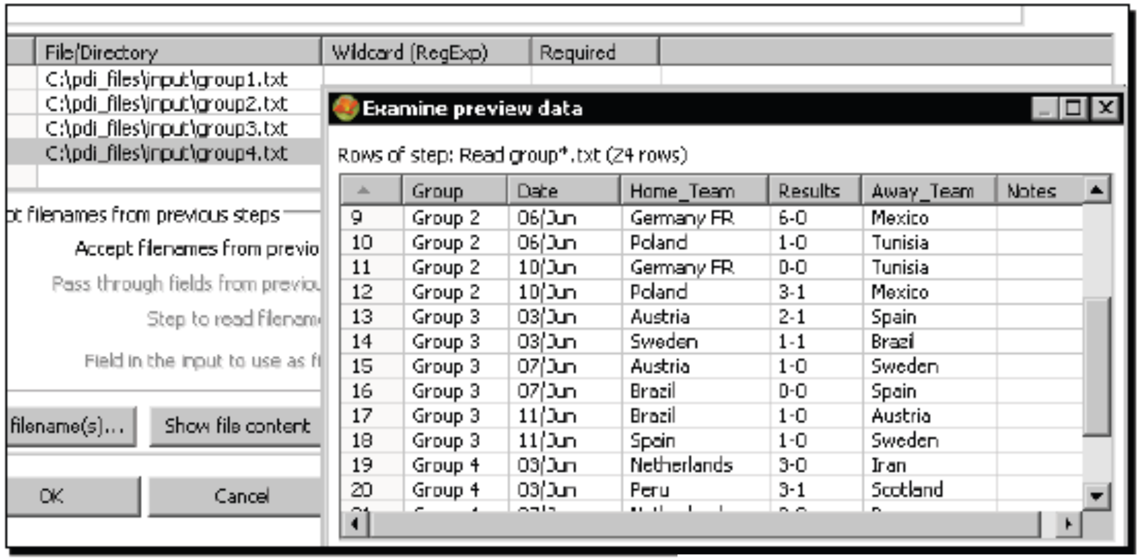



Examine preview data

Rows of step: Dummy (do nothing) (6 rows)

#.	Date	Home_Team	Results	Away_Team
1	02/Jun	Italy	2-1	France
2	02/Jun	Argentina	2-1	Hungary
3	06/Jun	Italy	3-1	Hungary
4	06/Jun	Argentina	2-1	France
5	10/Jun	France	3-1	Hungary
6	10/Jun	Italy	1-0	Argentina

Reading multiple files



File/Directory	Wildcard (RegExp)	Required
C:\pdi_files\input\group1.txt		
C:\pdi_files\input\group2.txt		
C:\pdi_files\input\group3.txt		
C:\pdi_files\input\group4.txt		

t filenames from previous steps

Accept filenames from previo

Pass through fields from previo

Step to read filenam

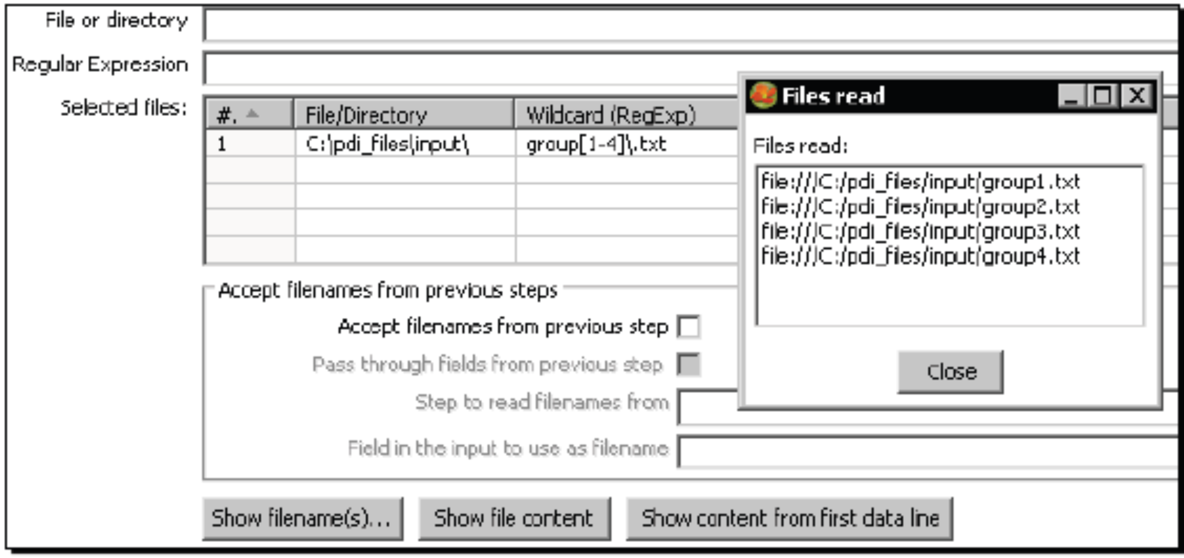
Field in the input to use as fi

Examine preview data

Rows of step: Read group*.txt (24 rows)

	Group	Date	Home_Team	Results	Away_Team	Notes
9	Group 2	06/Jun	Germany FR	6-0	Mexico	
10	Group 2	06/Jun	Poland	1-0	Tunisia	
11	Group 2	10/Jun	Germany FR	0-0	Tunisia	
12	Group 2	10/Jun	Poland	3-1	Mexico	
13	Group 3	03/Jun	Austria	2-1	Spain	
14	Group 3	03/Jun	Sweden	1-1	Brazil	
15	Group 3	07/Jun	Austria	1-0	Sweden	
16	Group 3	07/Jun	Brazil	0-0	Spain	
17	Group 3	11/Jun	Brazil	1-0	Austria	
18	Group 3	11/Jun	Spain	1-0	Sweden	
19	Group 4	03/Jun	Netherlands	3-0	Iran	
20	Group 4	03/Jun	Peru	3-1	Scotland	

Text file input



File or directory

Regular Expression

Selected files:

#.	File/Directory	Wildcard (RegExp)
1	C:\pdi_files\input\	group[1-4]\.txt

Accept filenames from previous steps

Accept filenames from previous step ☐

Pass through fields from previous step ☐

Step to read filenames from

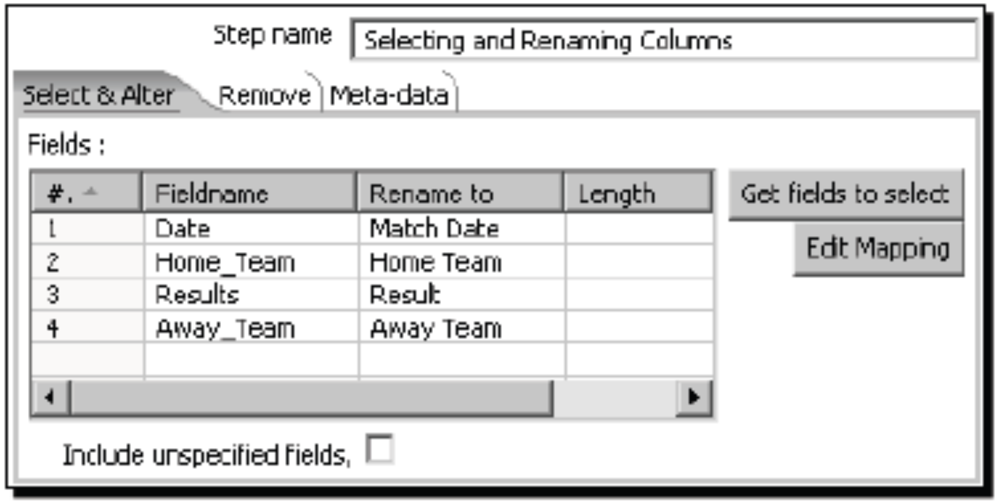
Field in the input to use as filename

Files read

Files read:

```
file:///C:/pdi_files/input/group1.txt
file:///C:/pdi_files/input/group2.txt
file:///C:/pdi_files/input/group3.txt
file:///C:/pdi_files/input/group4.txt
```

Sending data to files



Step name: Selecting and Renaming Columns

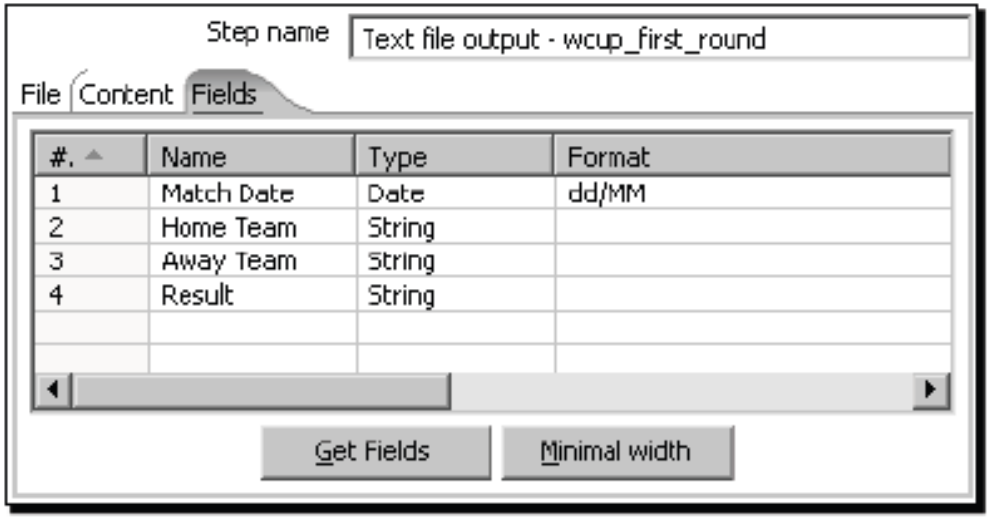
Fields :

#.	Fieldname	Rename to	Length
1	Date	Match Date	
2	Home_Team	Home Team	
3	Results	Result	
4	Away_Team	Away Team	

Include unspecified fields, ☐

Discard null column

Text file output

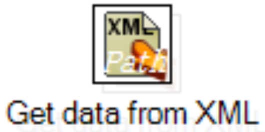


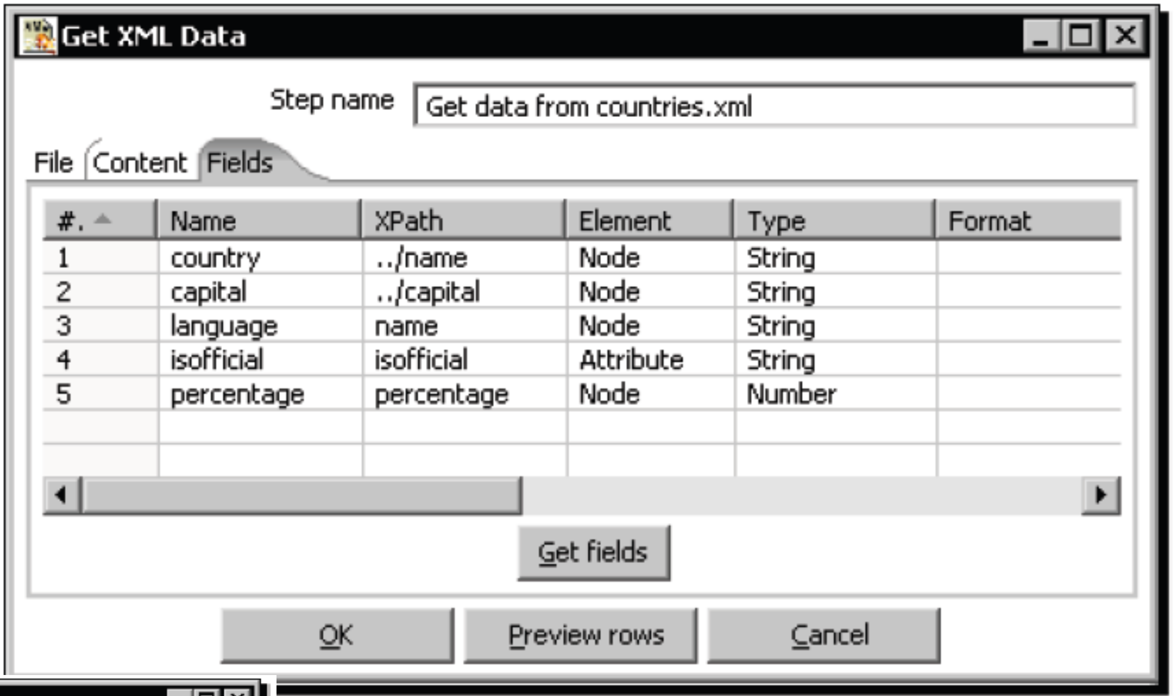
Step name: Text file output - wcup_first_round

#.	Name	Type	Format
1	Match Date	Date	dd/MM
2	Home Team	String	
3	Away Team	String	
4	Result	String	

Reading XML (countries.xml file)

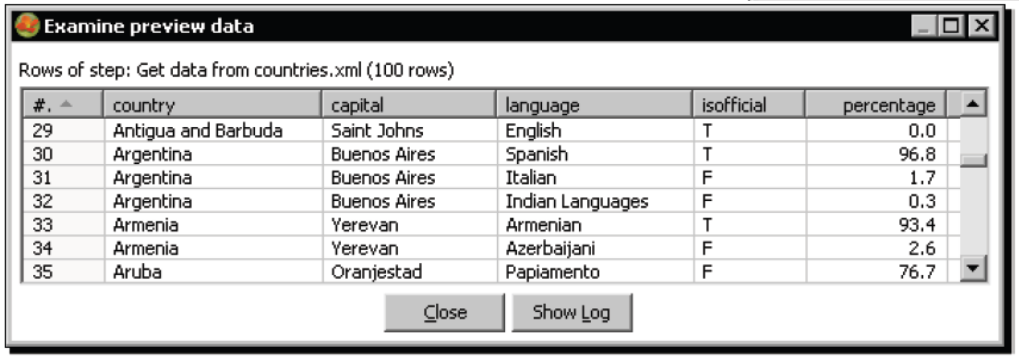
- In the Content tab, select **/world/country/language** for Loop XPath.





Step name: Get data from countries.xml

#.	Name	XPath	Element	Type	Format
1	country	../name	Node	String	
2	capital	../capital	Node	String	
3	language	name	Node	String	
4	isofficial	isofficial	Attribute	String	
5	percentage	percentage	Node	Number	

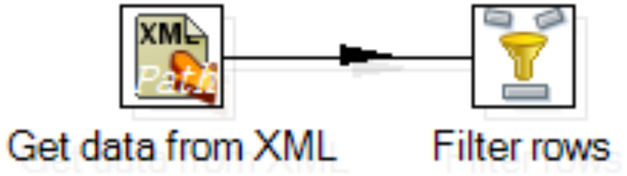


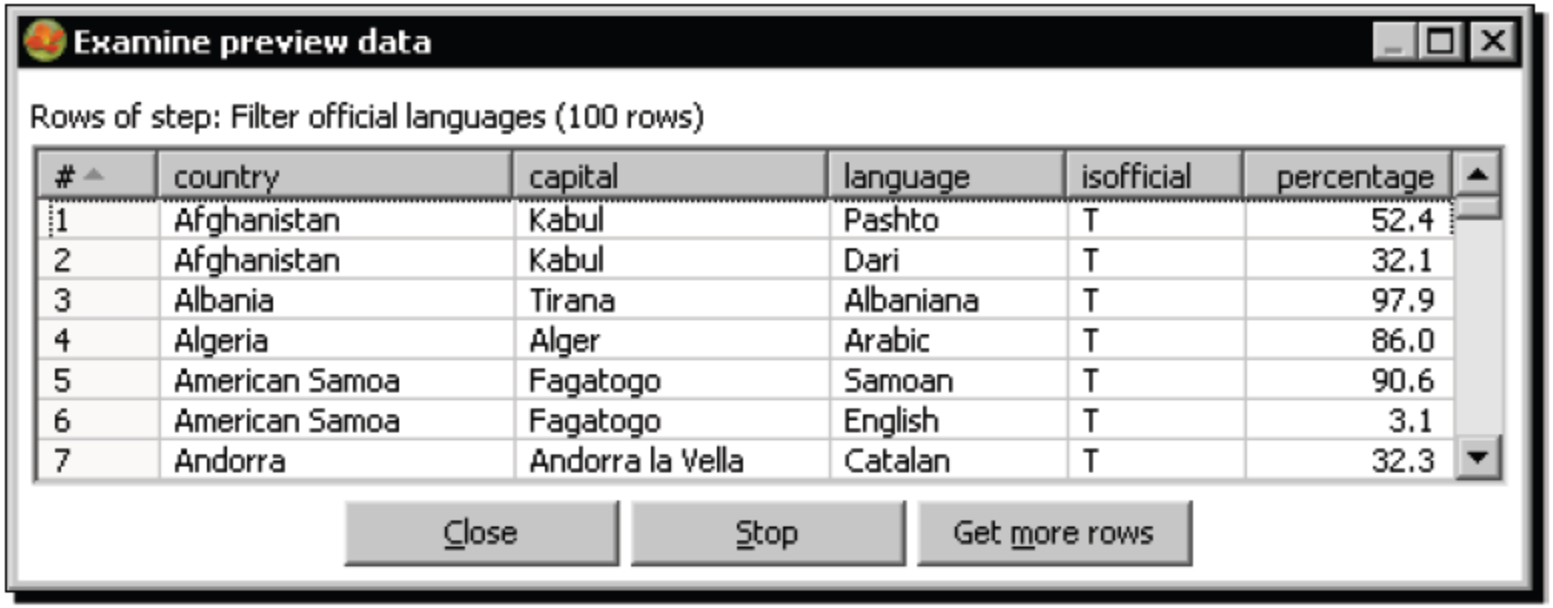
Rows of step: Get data from countries.xml (100 rows)

#.	country	capital	language	isofficial	percentage
29	Antigua and Barbuda	Saint Johns	English	T	0.0
30	Argentina	Buenos Aires	Spanish	T	96.8
31	Argentina	Buenos Aires	Italian	F	1.7
32	Argentina	Buenos Aires	Indian Languages	F	0.3
33	Armenia	Yerevan	Armenian	T	93.4
34	Armenia	Yerevan	Azerbaijani	F	2.6
35	Aruba	Oranjestad	Papiamento	F	76.7

Filter the rows

- Add a **Filter rows** step with the condition: **isofficial= T.**



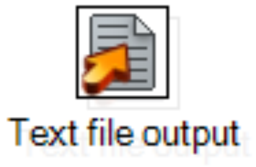


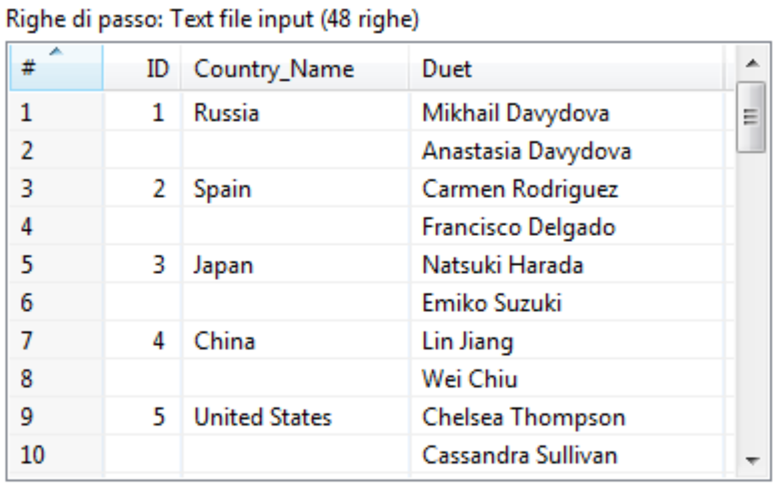
Rows of step: Filter official languages (100 rows)

#	country	capital	language	isofficial	percentage
1	Afghanistan	Kabul	Pashto	T	52.4
2	Afghanistan	Kabul	Dari	T	32.1
3	Albania	Tirana	Albaniana	T	97.9
4	Algeria	Alger	Arabic	T	86.0
5	American Samoa	Fagatogo	Samoan	T	90.6
6	American Samoa	Fagatogo	English	T	3.1
7	Andorra	Andorra la Vella	Catalan	T	32.3

Text file Input

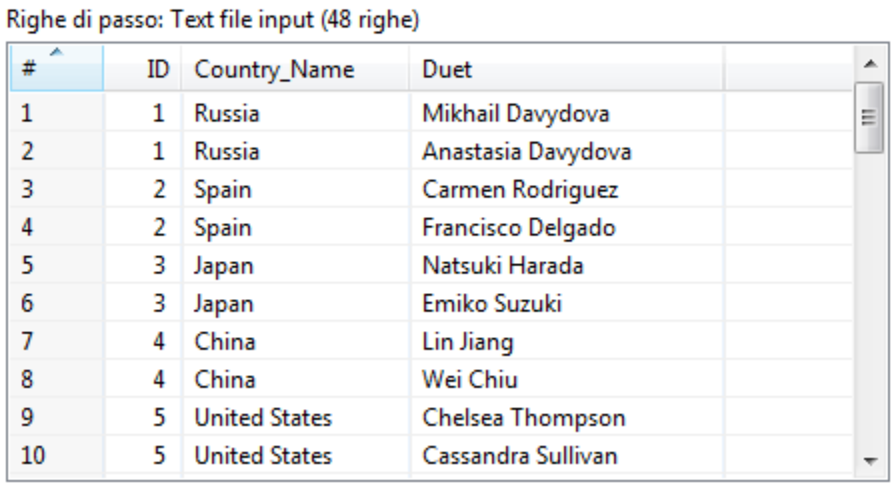
- The ID and country have values only in the first of the two lines for each country. In order to repeat the values in the second line use the flag **Repeat** in the **Fields** tab. Set it to **Y.**





Righe di passo: Text file input (48 righe)

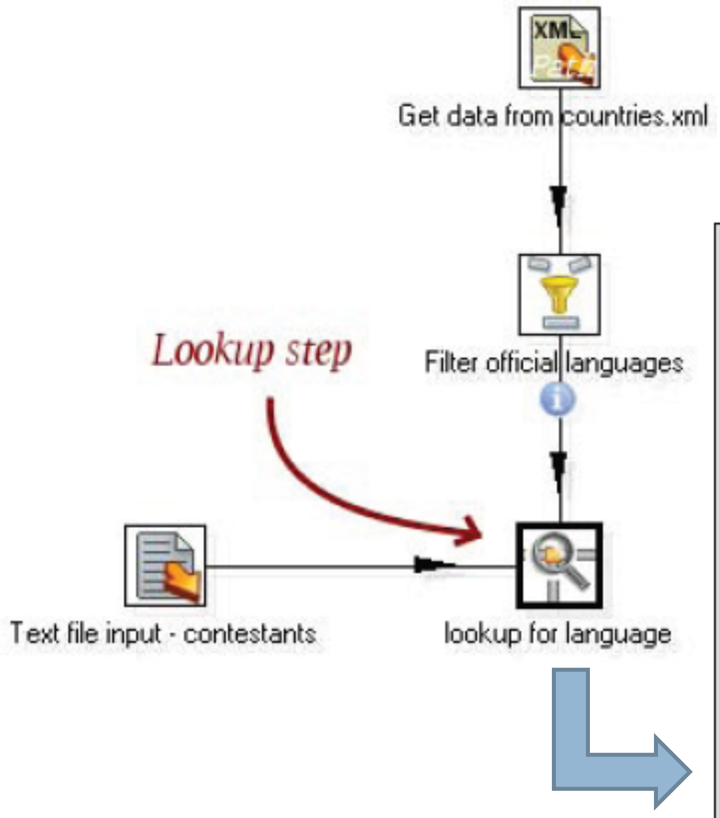
#	ID	Country_Name	Duet
1	1	Russia	Mikhail Davydova
2			Anastasia Davydova
3	2	Spain	Carmen Rodriguez
4			Francisco Delgado
5	3	Japan	Natsuki Harada
6			Emiko Suzuki
7	4	China	Lin Jiang
8			Wei Chiu
9	5	United States	Chelsea Thompson
10			Cassandra Sullivan

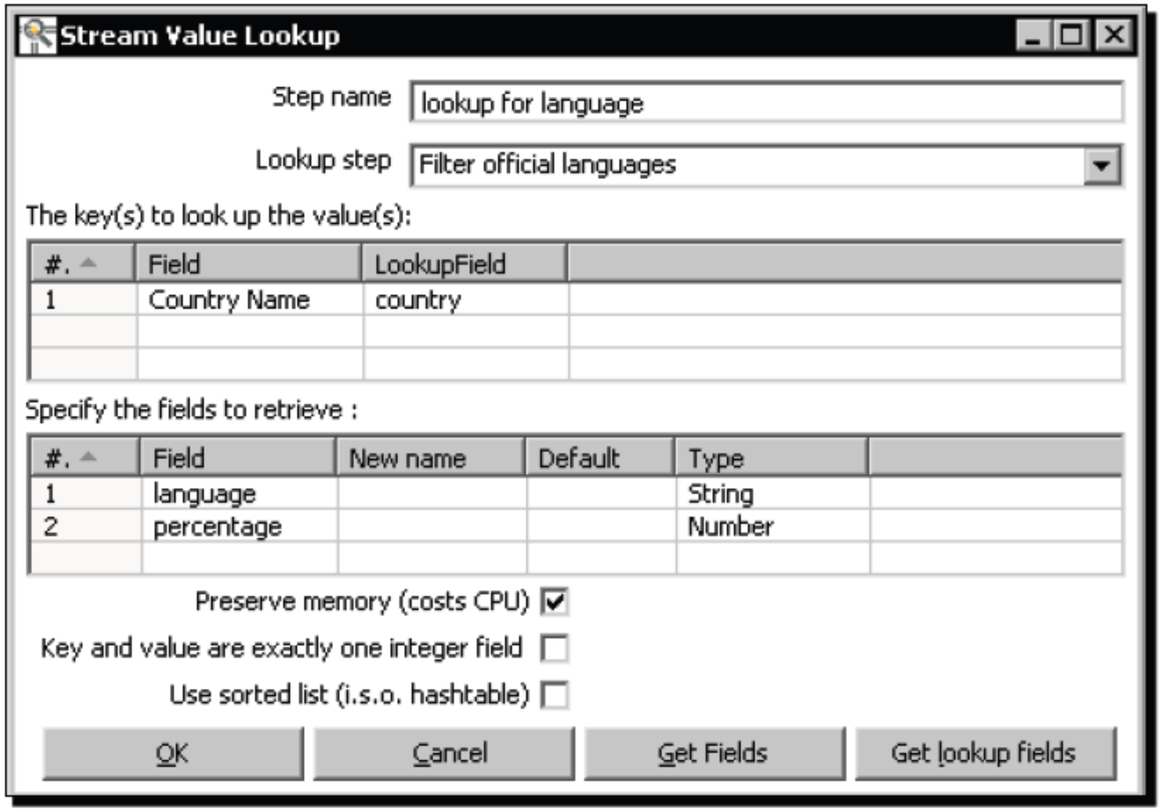


Righe di passo: Text file input (48 righe)

#	ID	Country_Name	Duet
1	1	Russia	Mikhail Davydova
2	1	Russia	Anastasia Davydova
3	2	Spain	Carmen Rodriguez
4	2	Spain	Francisco Delgado
5	3	Japan	Natsuki Harada
6	3	Japan	Emiko Suzuki
7	4	China	Lin Jiang
8	4	China	Wei Chiu
9	5	United States	Chelsea Thompson
10	5	United States	Cassandra Sullivan

Stream lookup


Get data from countries.xml
Filter official languages
Lookup step
Text file input - contestants
lookup for language



Step name: lookup for language

Lookup step: Filter official languages

The key(s) to look up the value(s):

#.	Field	LookupField
1	Country Name	country

Specify the fields to retrieve :

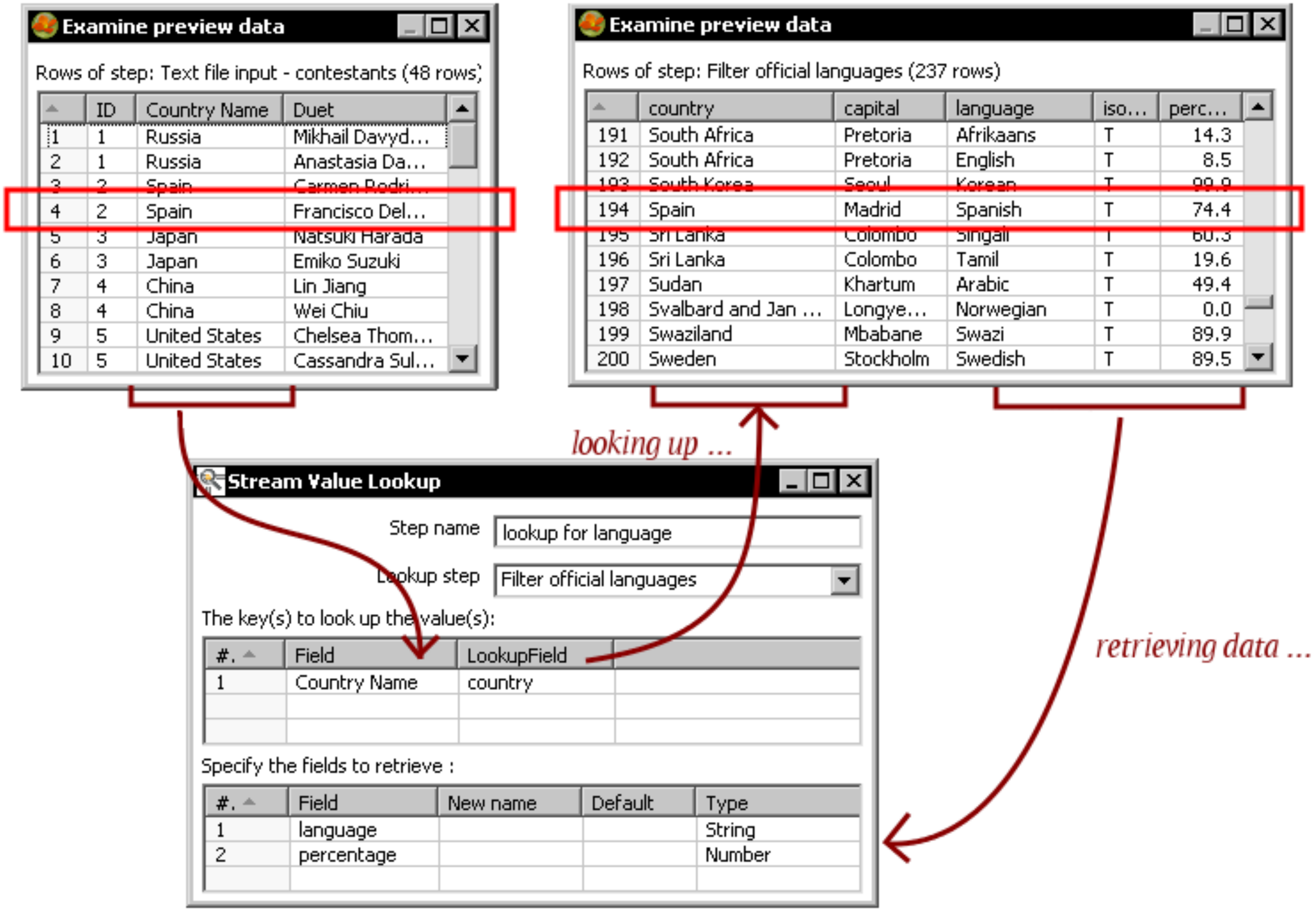
#.	Field	New name	Default	Type
1	language			String
2	percentage			Number

Preserve memory (costs CPU) ☑

Key and value are exactly one integer field ☐

Use sorted list (i.s.o. hashtable) ☐

Stream lookup



Examine preview data

Rows of step: Text file input - contestants (48 rows)

	ID	Country Name	Duet
1	1	Russia	Mikhail Davyd...
2	1	Russia	Anastasia Da...
3	2	Spain	Carmen Rodri...
4	2	Spain	Francisco Del...
5	3	Japan	Natsuki Harada
6	3	Japan	Emiko Suzuki
7	4	China	Lin Jiang
8	4	China	Wei Chiu
9	5	United States	Chelsea Thom...
10	5	United States	Cassandra Sul...

Examine preview data

Rows of step: Filter official languages (237 rows)

	country	capital	language	iso...	perc...
191	South Africa	Pretoria	Afrikaans	T	14.3
192	South Africa	Pretoria	English	T	8.5
193	South Korea	Seoul	Korean	T	99.9
194	Spain	Madrid	Spanish	T	74.4
195	Sri Lanka	Colombo	Singali	T	60.3
196	Sri Lanka	Colombo	Tamil	T	19.6
197	Sudan	Khartum	Arabic	T	49.4
198	Svalbard and Jan ...	Longye...	Norwegian	T	0.0
199	Swaziland	Mbabane	Swazi	T	89.9
200	Sweden	Stockholm	Swedish	T	89.5

Stream Value Lookup

Step name: lookup for language

Lookup step: Filter official languages

The key(s) to look up the value(s):

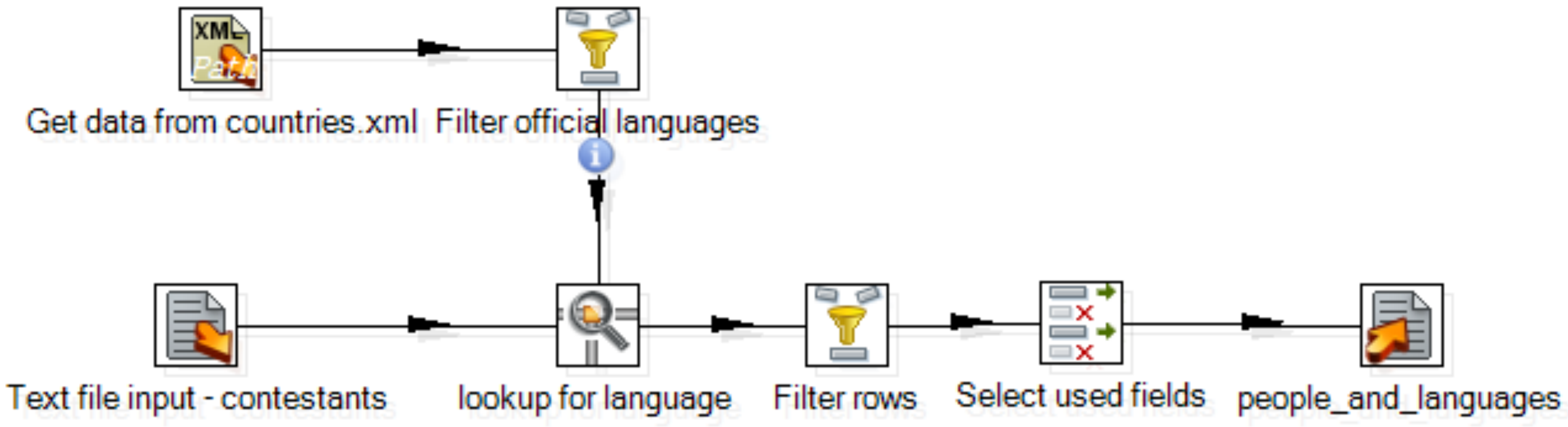
#.	Field	LookupField
1	Country Name	country

Specify the fields to retrieve :

#.	Field	New name	Default	Type
1	language			String
2	percentage			Number

Filter rows - Select values

- In the **Filter rows** step, type the condition **language IS NOT NULL**.
- By using a **Select values** step, rename the fields Duet, Country Name and language to Name, Country, and Language.


Get data from countries.xml
Filter official languages
Text file input - contestants
lookup for language
Filter rows
Select used fields
people_and_languages

Querying a database

1. Create a new transformation.
2. Select the **Design** view.
3. Expand the input category of steps and drag a **Table Input** step to the canvas.
4. Double-click the step.
5. Click on the **Get SQL select statement...** button. The database explorer window appears.
6. Expand the tables list and select `ORDERS`.
7. Click on **OK**.
8. PDI asks if you want to include the field names in the SQL. Answer **Yes**.

Querying a database (2)

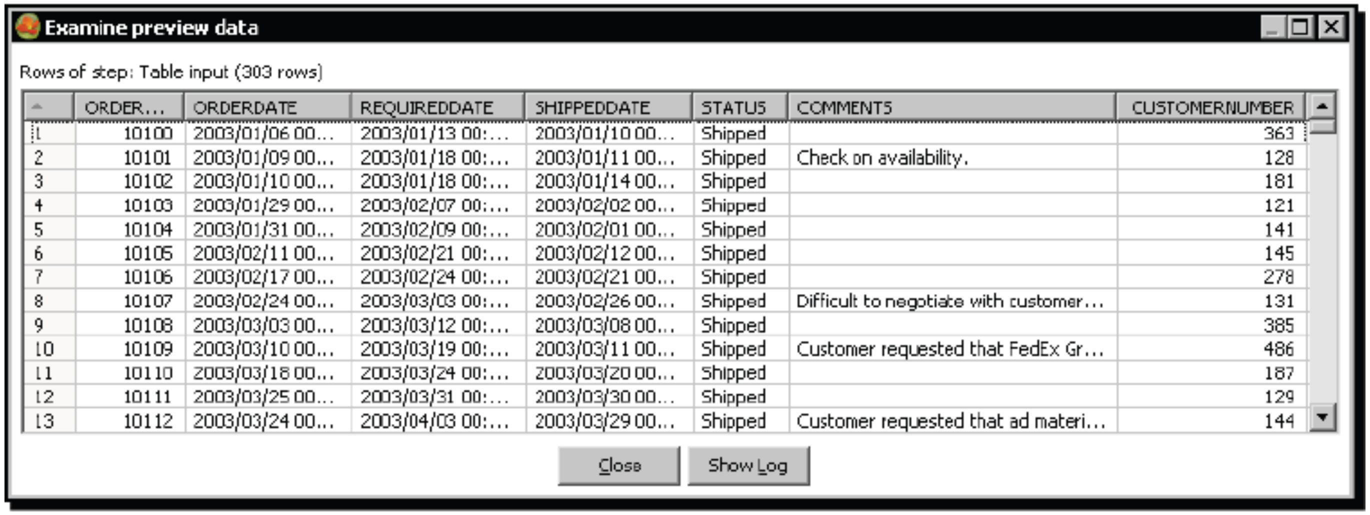
9. The SQL box gets filled with a SELECT SQL statement.

```
SELECT
  ORDERNUMBER
, ORDERDATE
, REQUIREDDATE
, SHIPPEDDATE
, STATUS
, COMMENTS
, CUSTOMERNUMBER
FROM ORDERS
```

10. At the end of the SQL statement, add the following clause:

```
WHERE STATUS = 'Shipped'
```

11. Click **Preview** and then **OK**. The following window appears:



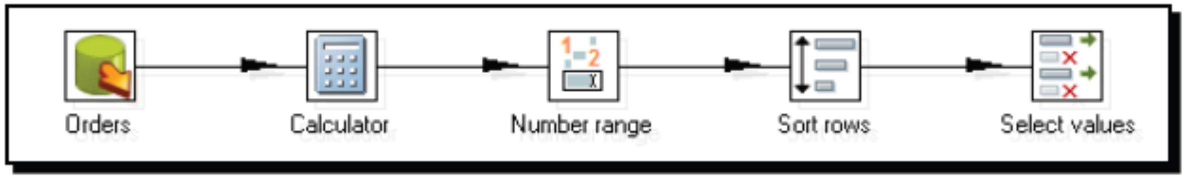
Rows of step: Table input (303 rows)

	ORDER...	ORDERDATE	REQUIREDDATE	SHIPPEDDATE	STATUS	COMMENTS	CUSTOMERNUMBER
1	10100	2003/01/06 00...	2003/01/13 00:...	2003/01/10 00...	Shipped		363
2	10101	2003/01/09 00...	2003/01/18 00:...	2003/01/11 00...	Shipped	Check on availability.	128
3	10102	2003/01/10 00...	2003/01/18 00:...	2003/01/14 00...	Shipped		181
4	10103	2003/01/29 00...	2003/02/07 00:...	2003/02/02 00...	Shipped		121
5	10104	2003/01/31 00...	2003/02/09 00:...	2003/02/01 00...	Shipped		141
6	10105	2003/02/11 00...	2003/02/21 00:...	2003/02/12 00...	Shipped		145
7	10106	2003/02/17 00...	2003/02/24 00:...	2003/02/21 00...	Shipped		278
8	10107	2003/02/24 00...	2003/03/03 00:...	2003/02/26 00...	Shipped	Difficult to negotiate with customer...	131
9	10108	2003/03/03 00...	2003/03/12 00:...	2003/03/08 00...	Shipped		385
10	10109	2003/03/10 00...	2003/03/19 00:...	2003/03/11 00...	Shipped	Customer requested that FedEx Gr...	486
11	10110	2003/03/18 00...	2003/03/24 00:...	2003/03/20 00...	Shipped		187
12	10111	2003/03/25 00...	2003/03/31 00:...	2003/03/30 00...	Shipped		129
13	10112	2003/03/24 00...	2003/04/03 00:...	2003/03/29 00...	Shipped	Customer requested that ad materi...	144

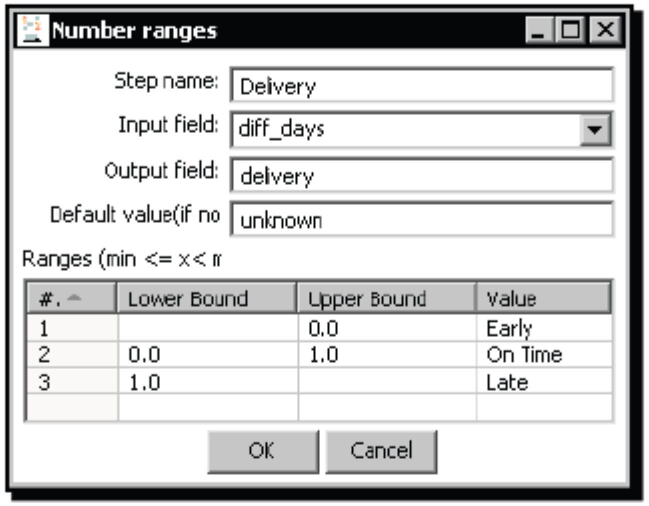
Querying a database (3)

12. Close the window and click **OK** to close the step configuration window.

13. After the **Table input** step add a **Calculator step**, a **Number Range** step, a **Sort** step, and a **Select values** step and link them as follows:



14. With the **Calculator** step, add an `Integer` field to calculate the difference between the shipped date and the required date. Use the calculation **Date A – Date B (in days)** and name the field `diff_days`. Use the **Number ranges** step to classify the delays in delivery.



Step name:	Delivery
Input field:	diff_days
Output field:	delivery
Default value(if no	unknown

Ranges (min <= x< m

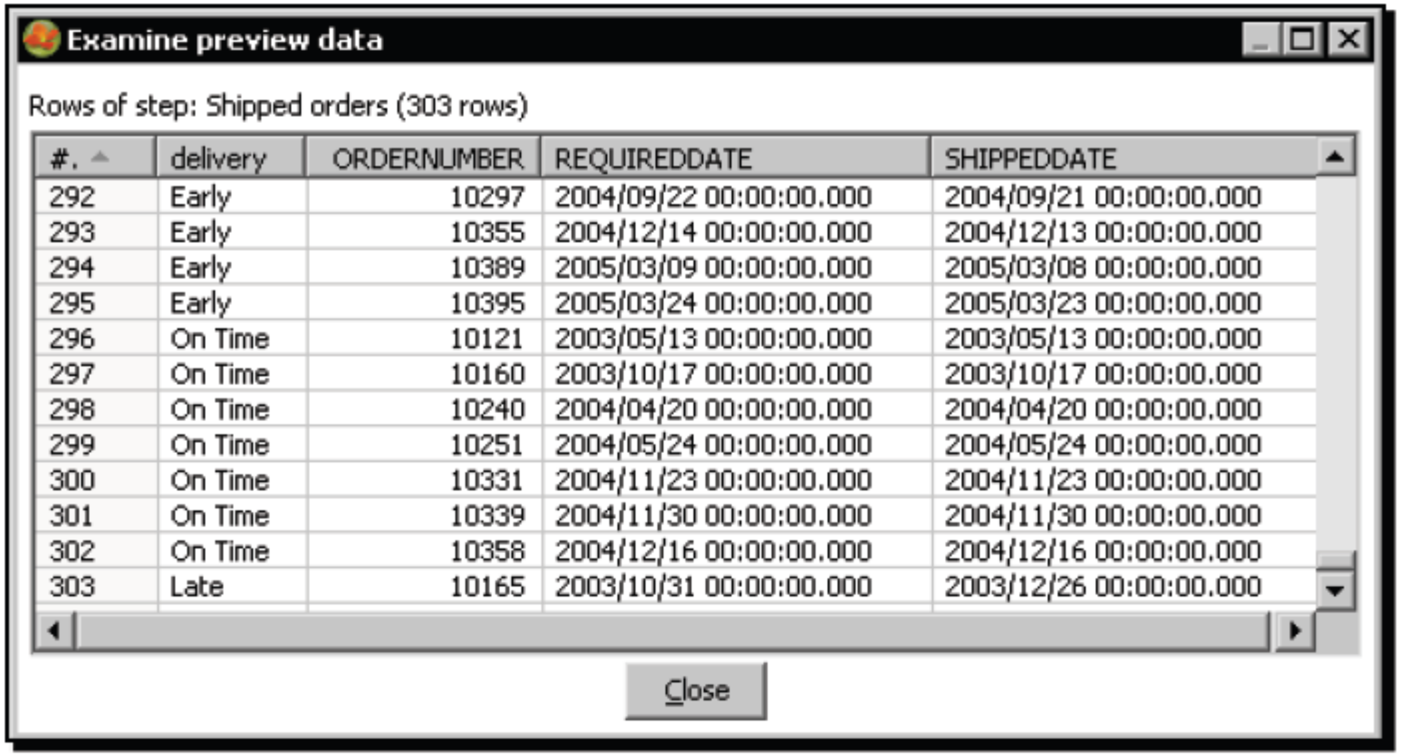
#.	Lower Bound	Upper Bound	Value
1		0.0	Early
2	0.0	1.0	On Time
3	1.0		Late

Querying a database (4)

15. Use the **Sort rows** step to sort the rows by the `diff_days` field.

16. Use the **Select values** step to select the `delivery`, `ORDERNUMBER`, `REQUIREDDATE`, and `SHIPPEDDATE` fields.

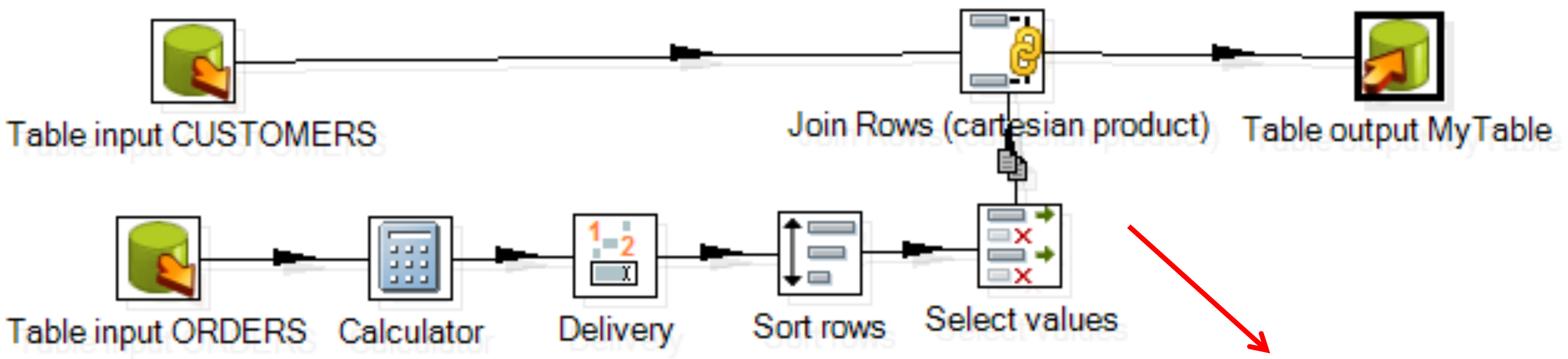
17. With the **Select values** step selected, do a preview. The following is how the final data will look:

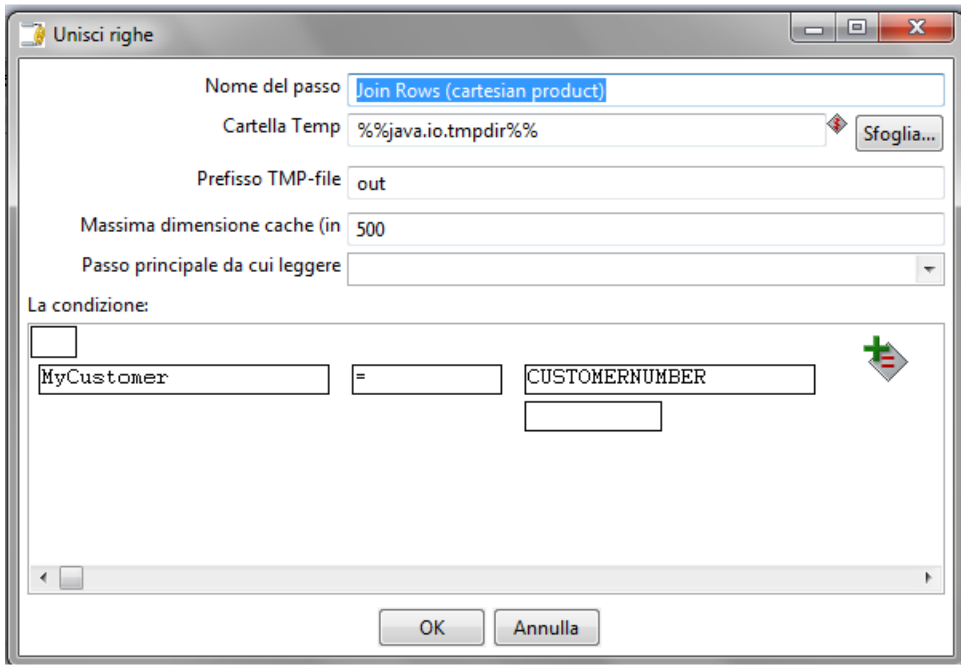


Rows of step: Shipped orders (303 rows)

#.	delivery	ORDERNUMBER	REQUIREDDATE	SHIPPEDDATE
292	Early	10297	2004/09/22 00:00:00.000	2004/09/21 00:00:00.000
293	Early	10355	2004/12/14 00:00:00.000	2004/12/13 00:00:00.000
294	Early	10389	2005/03/09 00:00:00.000	2005/03/08 00:00:00.000
295	Early	10395	2005/03/24 00:00:00.000	2005/03/23 00:00:00.000
296	On Time	10121	2003/05/13 00:00:00.000	2003/05/13 00:00:00.000
297	On Time	10160	2003/10/17 00:00:00.000	2003/10/17 00:00:00.000
298	On Time	10240	2004/04/20 00:00:00.000	2004/04/20 00:00:00.000
299	On Time	10251	2004/05/24 00:00:00.000	2004/05/24 00:00:00.000
300	On Time	10331	2004/11/23 00:00:00.000	2004/11/23 00:00:00.000
301	On Time	10339	2004/11/30 00:00:00.000	2004/11/30 00:00:00.000
302	On Time	10358	2004/12/16 00:00:00.000	2004/12/16 00:00:00.000
303	Late	10165	2003/10/31 00:00:00.000	2003/12/26 00:00:00.000

Saving into a database

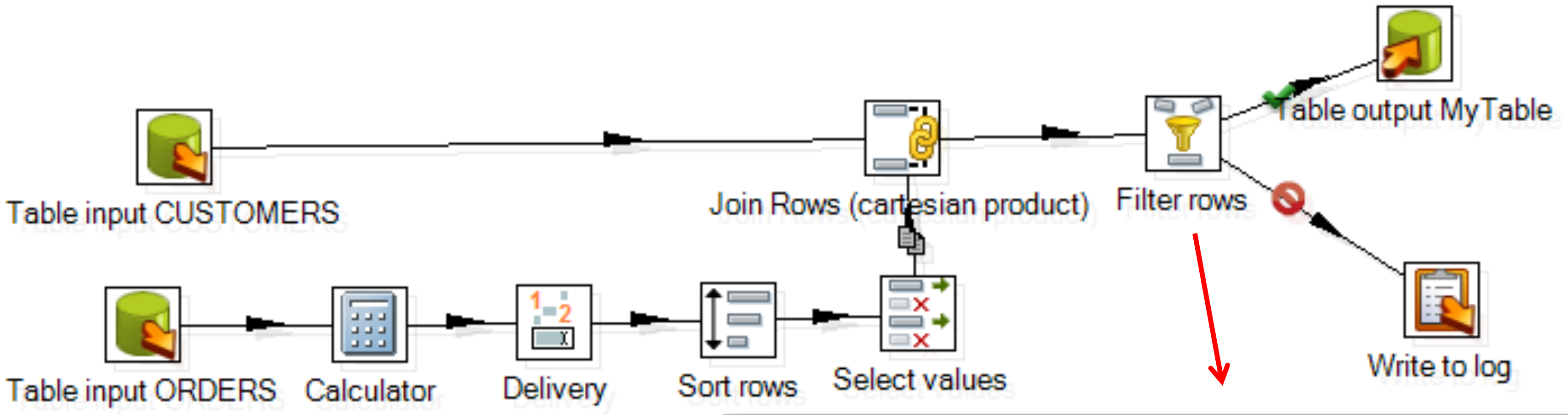


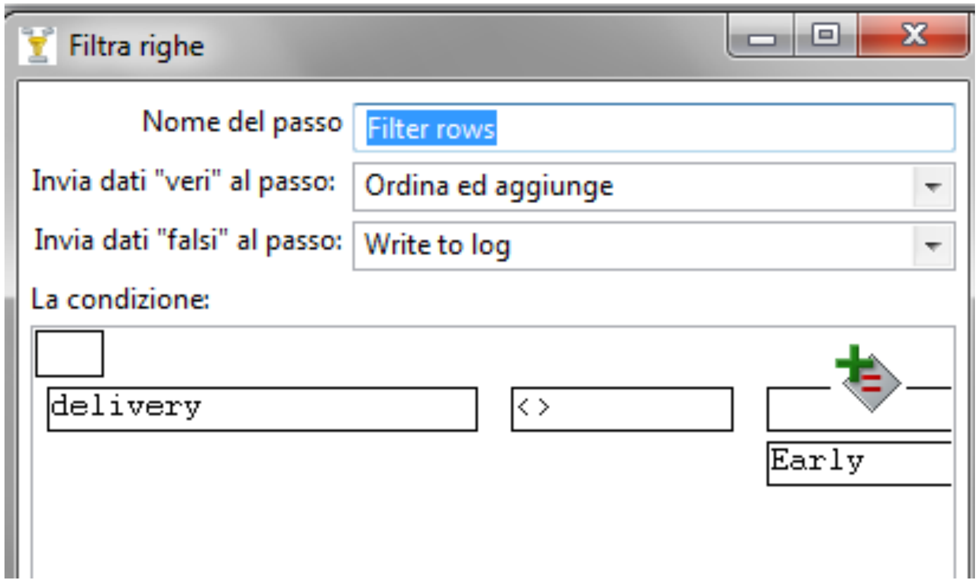


Nome del passo: Join Rows (cartesian product)
Cartella Temp: %%java.io.tmpdir%%
Prefisso TMP-file: out
Massima dimensione cache (in: 500
Passo principale da cui leggere:
La condizione:

```
MyCustomer    =    CUSTOMERNUMBER
```

Saving and logging




Filtra righe

Nome del passo: Filter rows

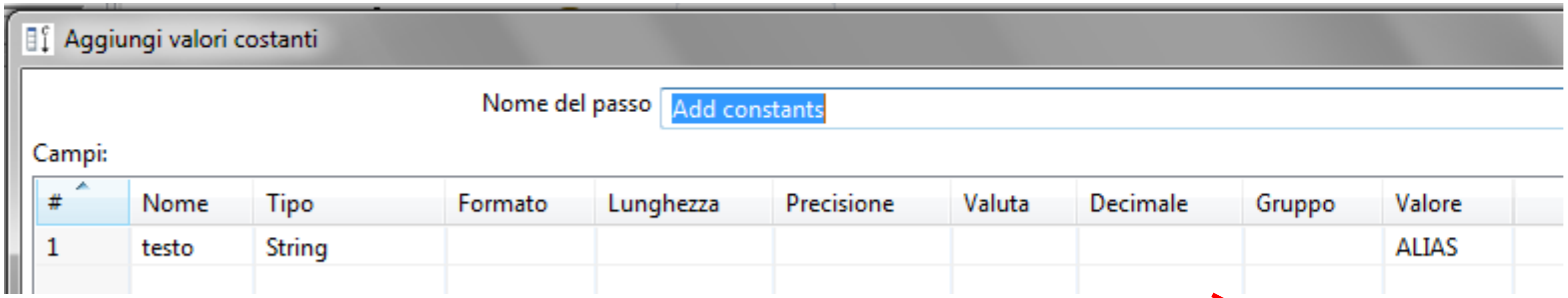
Invia dati "veri" al passo: Ordina ed aggiunge

Invia dati "falsi" al passo: Write to log

La condizione:

delivery <> Early

Saving and logging (2)

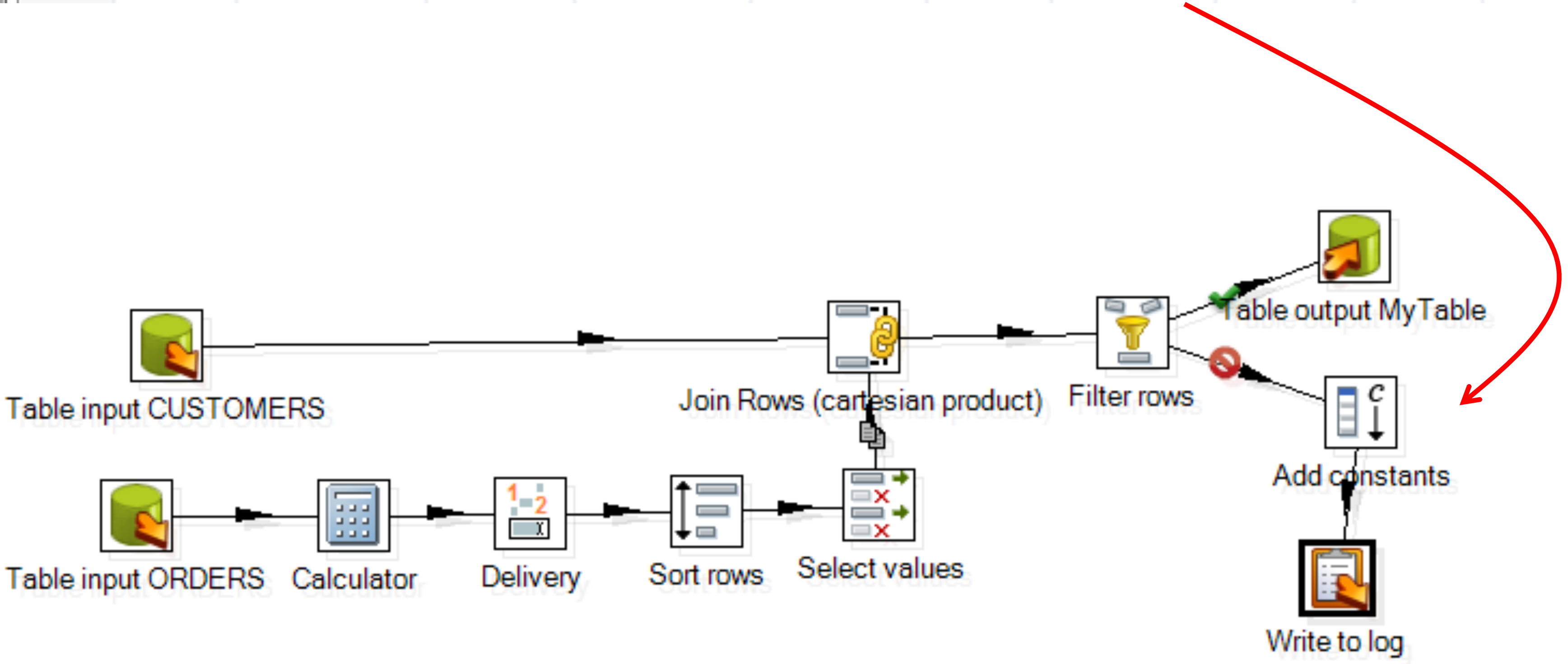


Aggiungi valori costanti

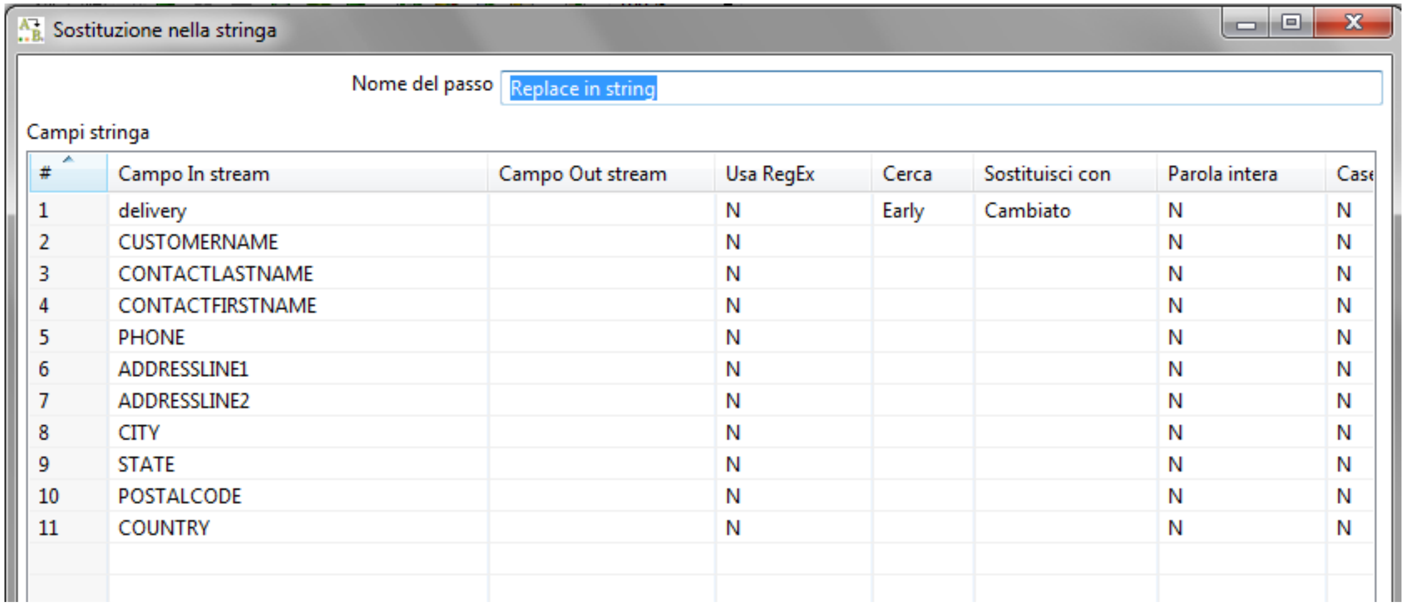
Nome del passo: Add constants

Campi:

#	Nome	Tipo	Formato	Lunghezza	Precisione	Valuta	Decimale	Gruppo	Valore
1	testo	String							ALIAS



Replacing

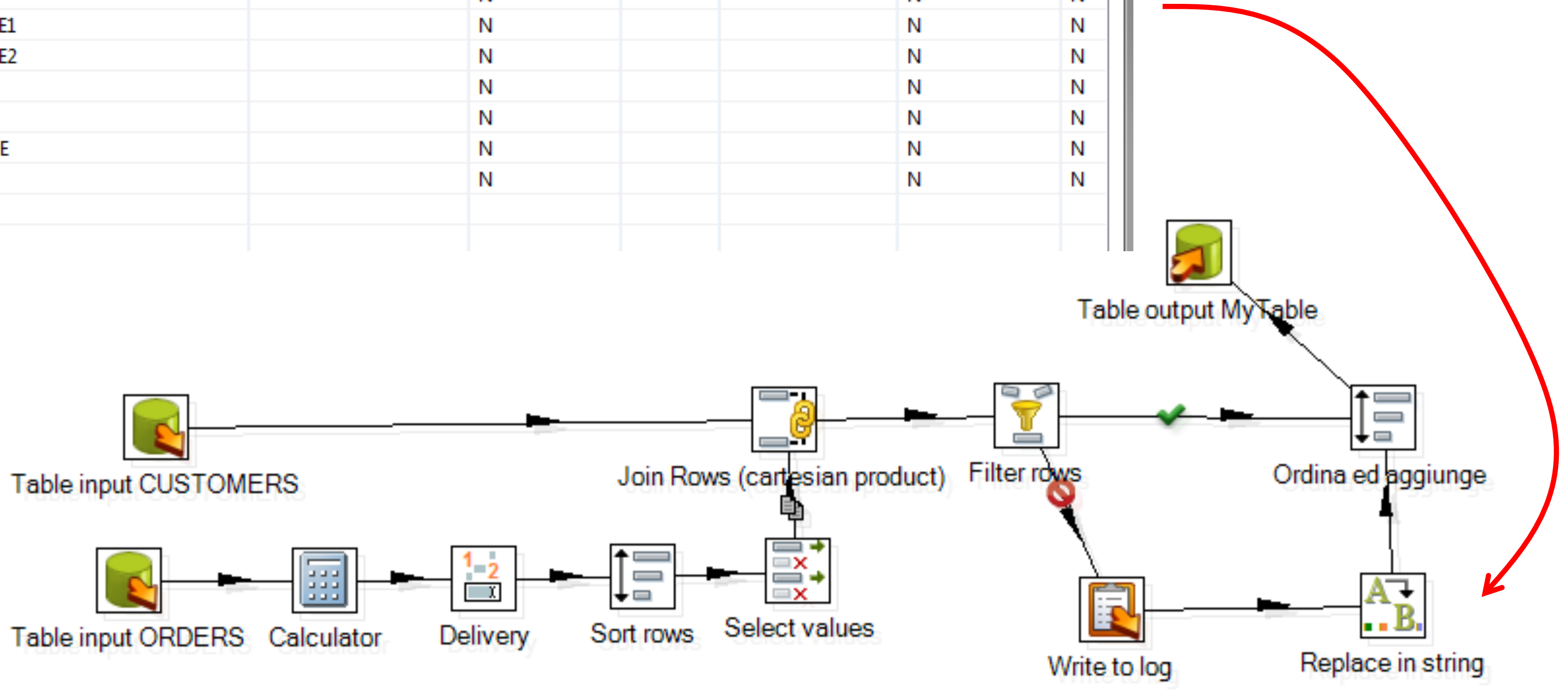


Sostituzione nella stringa

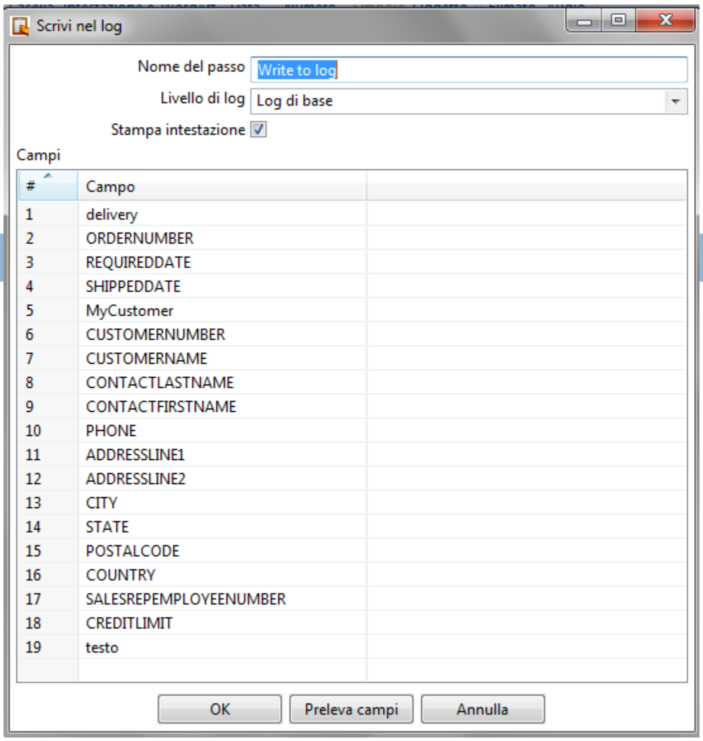
Nome del passo: Replace in string

Campi stringa

#	Campo In stream	Campo Out stream	Usa RegEx	Cerca	Sostituisci con	Parola intera	Case
1	delivery		N	Early	Cambiato	N	N
2	CUSTOMERNAME		N			N	N
3	CONTACTLASTNAME		N			N	N
4	CONTACTFIRSTNAME		N			N	N
5	PHONE		N			N	N
6	ADDRESSLINE1		N			N	N
7	ADDRESSLINE2		N			N	N
8	CITY		N			N	N
9	STATE		N			N	N
10	POSTALCODE		N			N	N
11	COUNTRY		N			N	N



Adding and removing a costant



Scrivi nel log

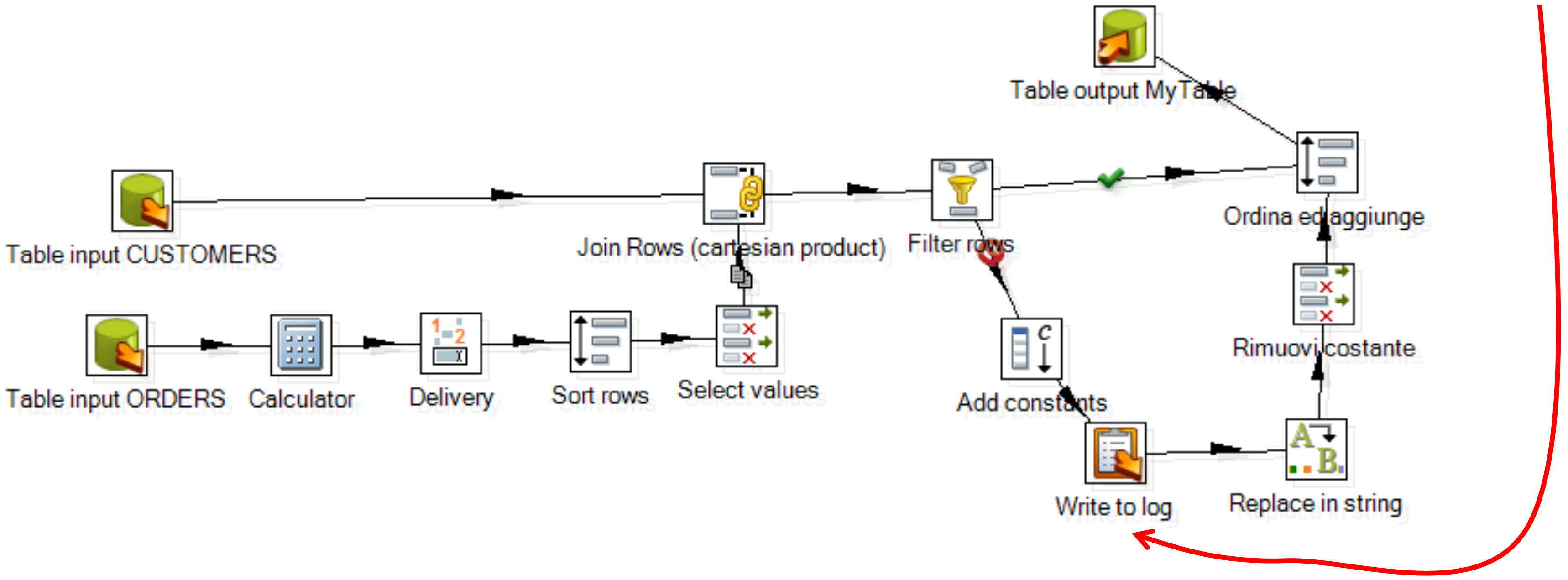
Nome del passo: Write to log

Livello di log: Log di base

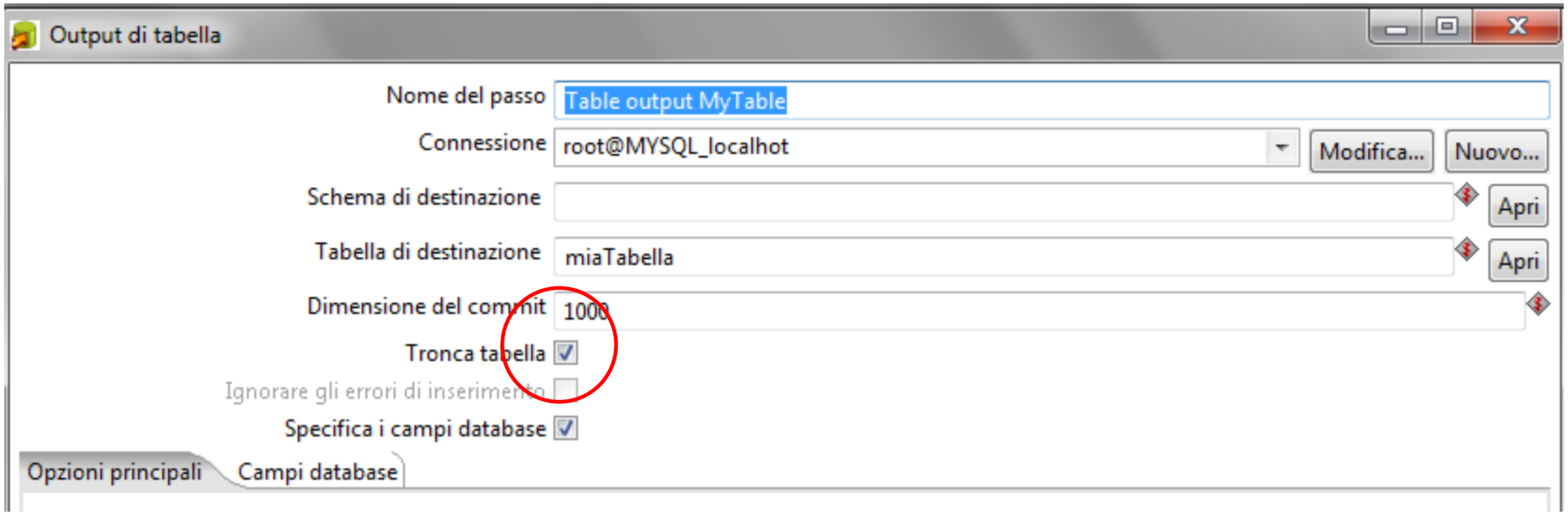
Stampa intestazione: ✓

Campi

#	Campo
1	delivery
2	ORDERNUMBER
3	REQUIREDDATE
4	SHIPPEDDATE
5	MyCustomer
6	CUSTOMERNUMBER
7	CUSTOMERNAME
8	CONTACTLASTNAME
9	CONTACTFIRSTNAME
10	PHONE
11	ADDRESSLINE1
12	ADDRESSLINE2
13	CITY
14	STATE
15	POSTALCODE
16	COUNTRY
17	SALESREPEMPLOYEENUMBER
18	CREDITLIMIT
19	testo



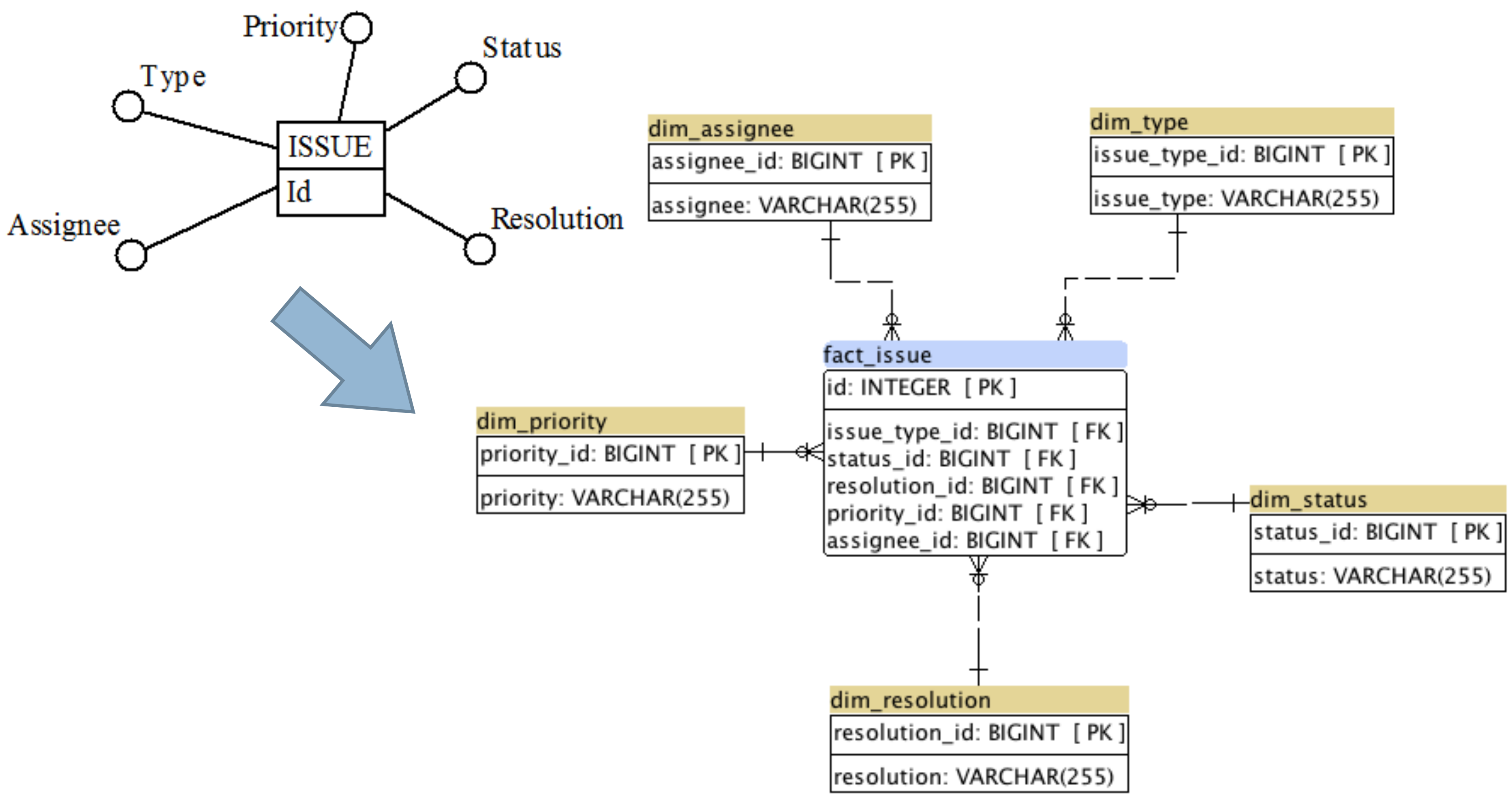
Deleting a table



Output di tabella

Nome del passo	Table output MyTable
Connessione	root@MYSQL_localhot
Schema di destinazione	
Tabella di destinazione	miaTabella
Dimensione del commit	1000
Tronca tabella	✓
Ignorare gli errori di inserimento	
Specifica i campi database	✓

A Basic Mondrian Cube (dfm → star schema)



The relational db schema
(execute this script in MySQL)

```
DROP DATABASE IF EXISTS sampleissue;
CREATE DATABASE sampleissue;
USE sampleissue;

-- type dimension
CREATE TABLE dim_type (
    issue_type_id BIGINT NOT NULL PRIMARY KEY,
    issue_type VARCHAR(255)
);
CREATE UNIQUE INDEX idx_dim_type_pk ON dim_type (issue_type_id);
CREATE INDEX idx_dim_type_lookup ON dim_type (issue_type);
-- ...
-- fact table

CREATE TABLE fact_issue (
    assignee_id BIGINT,
    issue_type_id BIGINT,
    priority_id BIGINT,
    status_id BIGINT,
    resolution_id BIGINT,
    id INT primary key
);
```

… *(this script is not complete)*

Building a Star Schema Cube

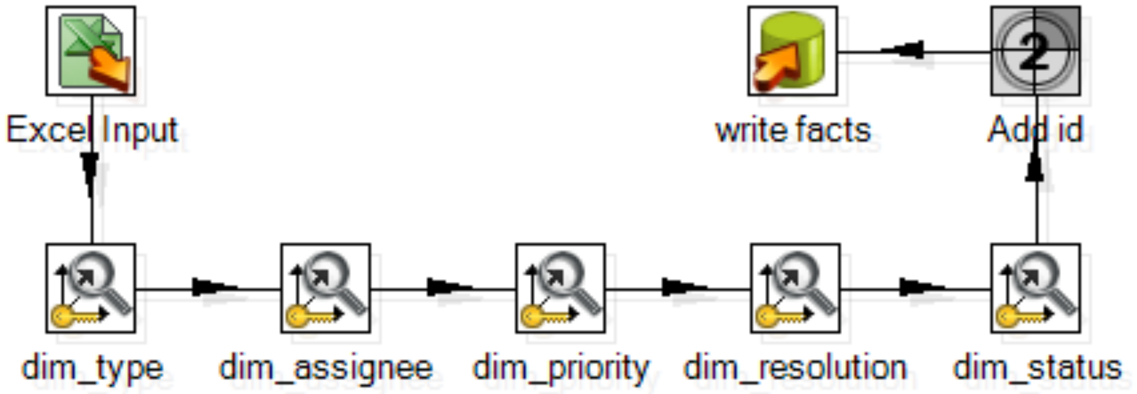
- For each dimension field it is necessary to store all possible values in a dimension table also generating an artificial key.
- This key should be used to reference the dimension values in the fact table.
- We build the cube using the Kettle ETL tool.
- *The MySQL DBMS should be active and accessible.*

Kettle transformation

- The transformation is going to make use of the **“Combination lookup/update”** step from the **“Data Warehouse”** section.
  - It does exactly what is required to fill the dimension tables and create the artificial keys.
- The step is configured with a table, the name of the artificial key field, and the row stream fields that make up the dimension.
- When a row passes this step the fields are looked up in the dimension table.
  - If a match is found, the corresponding key is added to the row stream.
  - If there is no match, the step creates a matching entry in the dimension table and puts the (newly generated) corresponding key to the row stream.

The trasformation

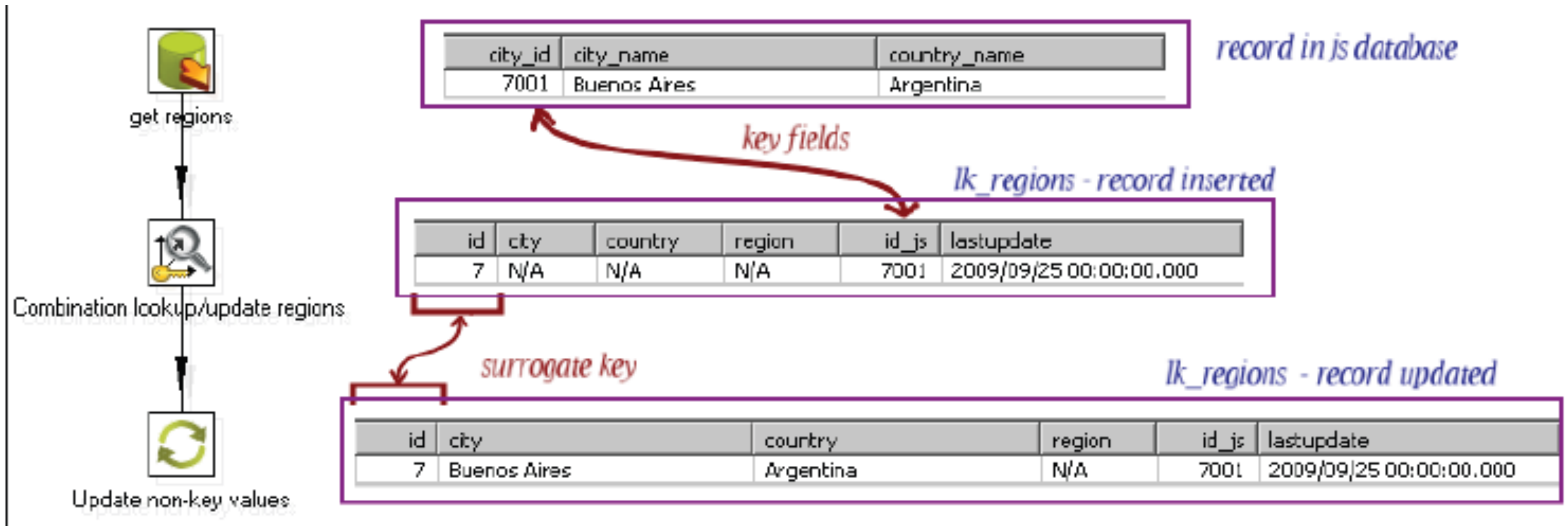
- So whenever a row passes a **“Combination lookup/update”** step, it ensures that there is a row with the dimension fields in the dimension table and puts the corresponding key to the row stream.
  - Exactly what is required to build a star schema cube.
- The following transformation recreates the cube using the Excel sheet as input.
  - It uses dimension tables.
  - For each dimension table there is a corresponding dimension step which is responsible for filling it.
  - At the end of the process the fact table is written.



Combination lookup/update step

- The Combination lookup/update, looks in the dimension table for a record that matches the key fields you put in the upper grid in the settings window.
  - If the combination exists, the step returns the surrogate key of the found record.
  - If it doesn't exist, the step generates a new surrogate key and inserts a row with the key fields and the generated surrogate key.
  - In any case, the surrogate key is added to the output stream.

Combination lookup/update step (2)

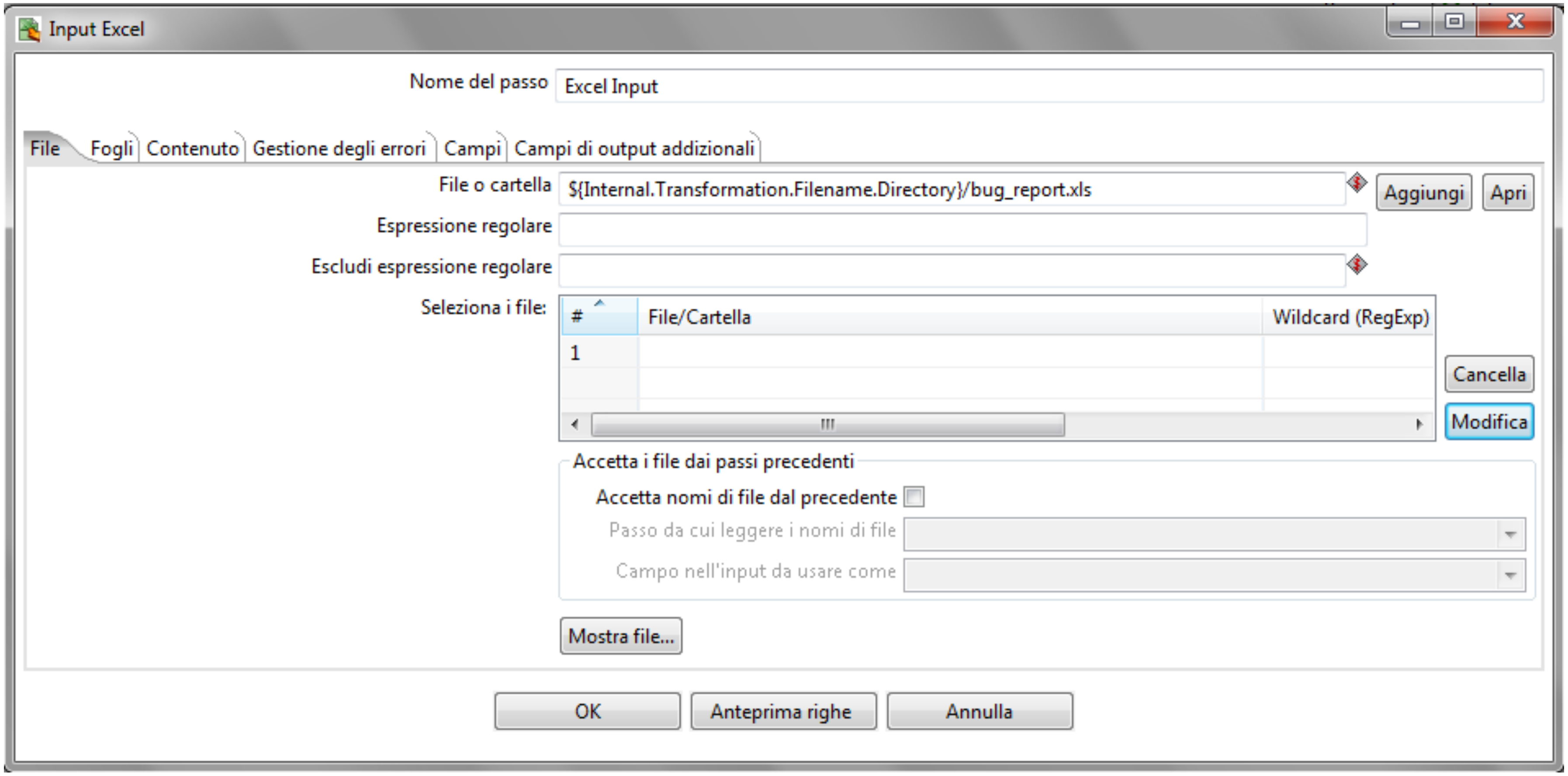

get regions
Combination lookup/update regions
Update non-key values
record in js database
key fields
lk_regions - record inserted
surrogate key
lk_regions - record updated

city_id	city_name	country_name
7001	Buenos Aires	Argentina

id	city	country	region	id_js	lastupdate
7	N/A	N/A	N/A	7001	2009/09/25 00:00:00.000

id	city	country	region	id_js	lastupdate
7	Buenos Aires	Argentina	N/A	7001	2009/09/25 00:00:00.000

Input Excel



Input Excel

Nome del passo: Excel Input

File | Fogli | Contenuto | Gestione degli errori | Campi | Campi di output addizionali

File o cartella: ${Internal.Transformation.Filename.Directory}/bug_report.xls

Espressione regolare:

Escludi espressione regolare:

Seleziona i file:

#	File/Cartella	Wildcard (RegExp)
1		

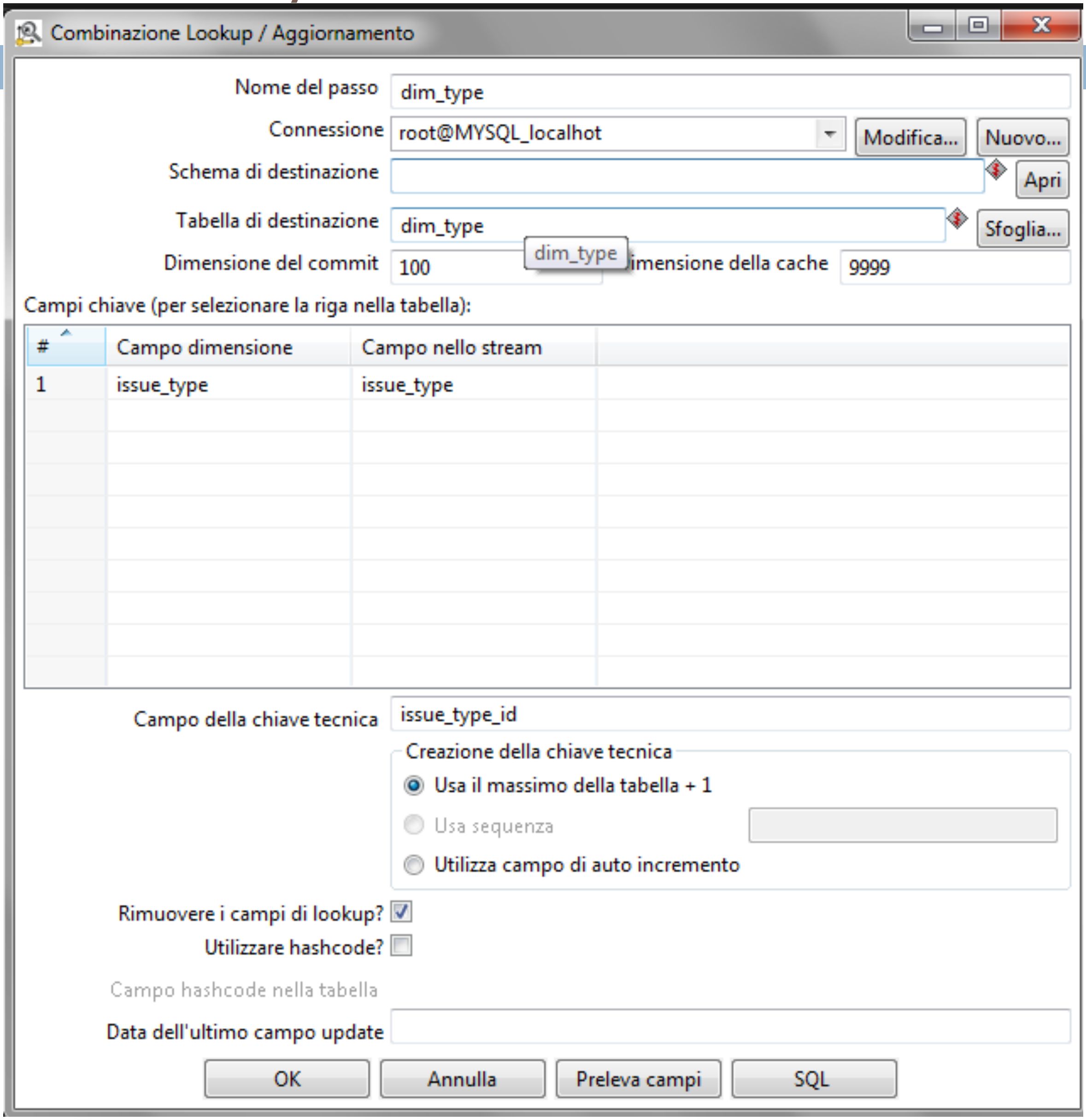
Accetta i file dai passi precedenti

Accetta nomi di file dal precedente ☐

Passo da cui leggere i nomi di file:

Campo nell'input da usare come:

Combination Lookup/update



Combinazione Lookup / Aggiornamento

Nome del passo	dim_type
Connessione	root@MYSQL_localhot
Schema di destinazione	
Tabella di destinazione	dim_type
Dimensione del commit	100
Dimensione della cache	9999

Campi chiave (per selezionare la riga nella tabella):

#	Campo dimensione	Campo nello stream
1	issue_type	issue_type

Campo della chiave tecnica: issue_type_id

Creazione della chiave tecnica

● Usa il massimo della tabella + 1

○ Usa sequenza

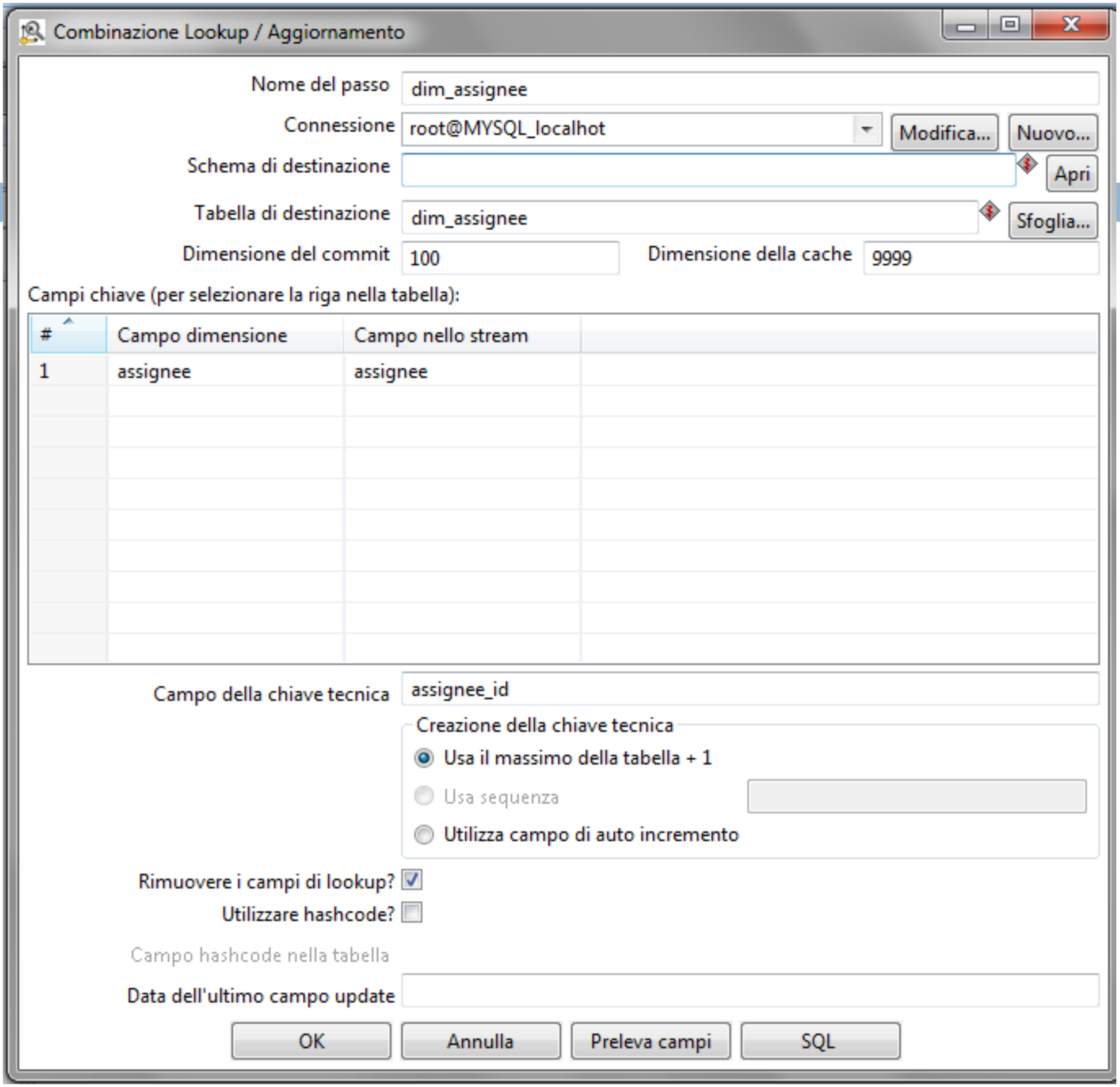
○ Utilizza campo di auto incremento

Rimuovere i campi di lookup? ☑

Utilizzare hashcode? ☐

Campo hashcode nella tabella

Data dell'ultimo campo update



Combinazione Lookup / Aggiornamento

Nome del passo	dim_assignee
Connessione	root@MYSQL_localhot
Schema di destinazione	
Tabella di destinazione	dim_assignee
Dimensione del commit	100
Dimensione della cache	9999

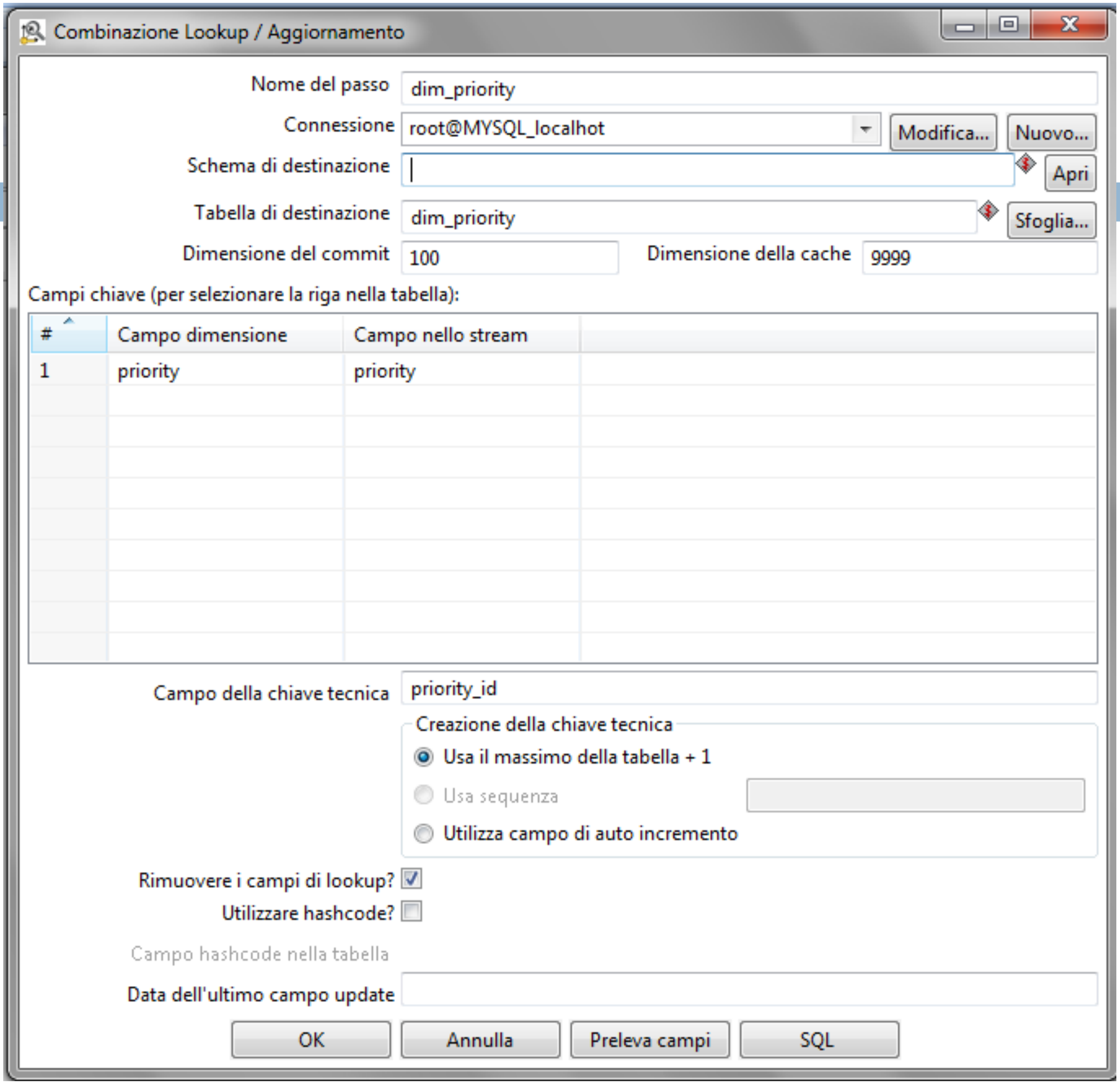
Campi chiave (per selezionare la riga nella tabella):

#	Campo dimensione	Campo nello stream
1	assignee	assignee

Campo della chiave tecnica	assignee_id

Creazione della chiave tecnica
- [✓ Usa il massimo della tabella + 1]
- Usa sequenza
- Utilizza campo di auto incremento

Rimuovere i campi di lookup?	✓
Utilizzare hashcode?	☐
Campo hashcode nella tabella	
Data dell'ultimo campo update	



Nome del passo: dim_priority

Connessione: root@MYSQL_localhot

Schema di destinazione:

Tabella di destinazione: dim_priority

Dimensione del commit: 100

Dimensione della cache: 9999

Campi chiave (per selezionare la riga nella tabella):

#	Campo dimensione	Campo nello stream
1	priority	priority

Campo della chiave tecnica: priority_id

Creazione della chiave tecnica

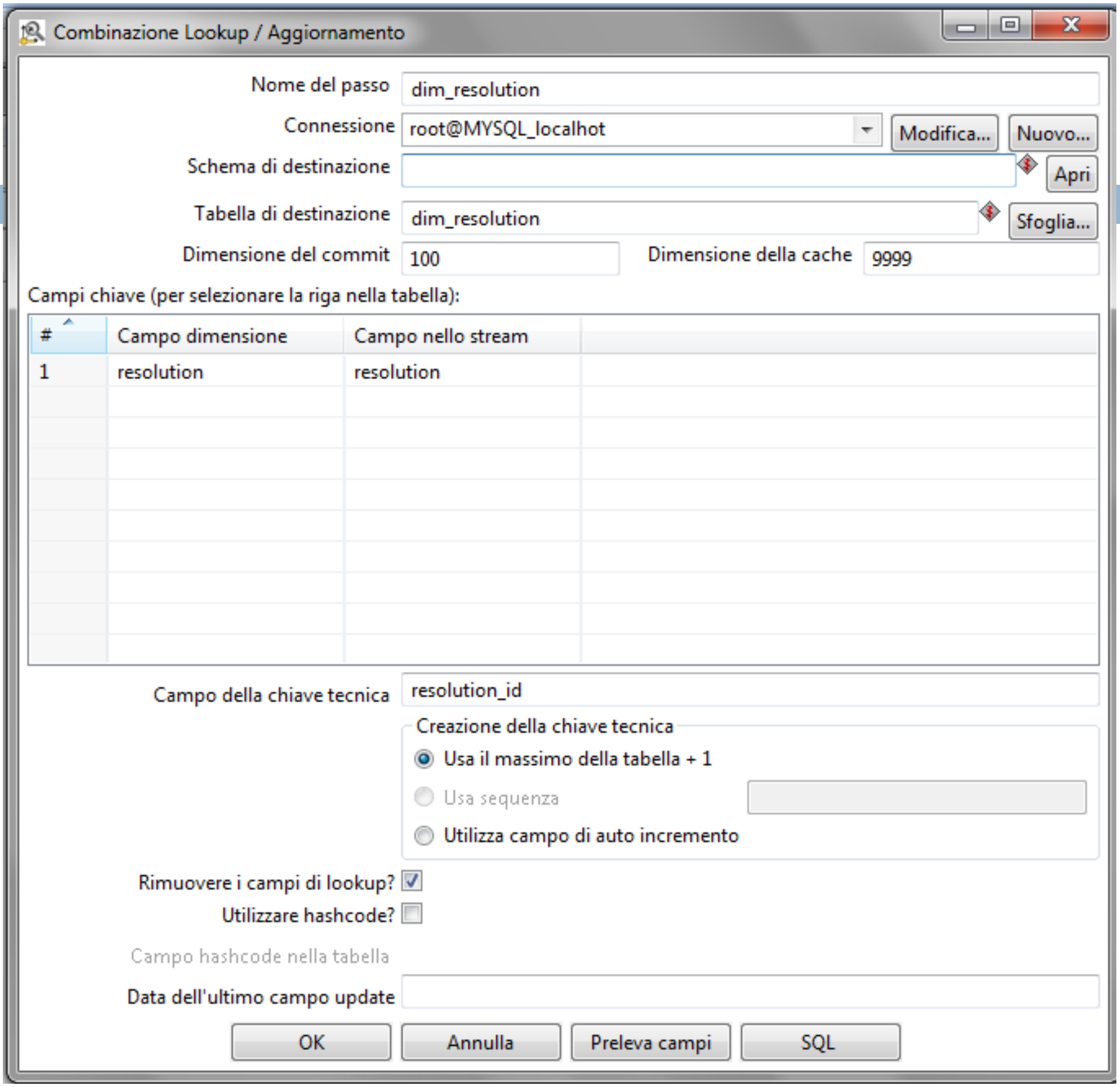
- [✓ Usa il massimo della tabella + 1]
- Usa sequenza
- Utilizza campo di auto incremento

Rimuovere i campi di lookup? [✓]

Utilizzare hashcode? [ ]

Campo hashcode nella tabella

Data dell'ultimo campo update:



Combinazione Lookup / Aggiornamento

Nome del passo	dim_resolution
Connessione	root@MYSQL_localhot
Schema di destinazione	
Tabella di destinazione	dim_resolution
Dimensione del commit	100
Dimensione della cache	9999

Campi chiave (per selezionare la riga nella tabella):

#	Campo dimensione	Campo nello stream
1	resolution	resolution

Campo della chiave tecnica: resolution_id

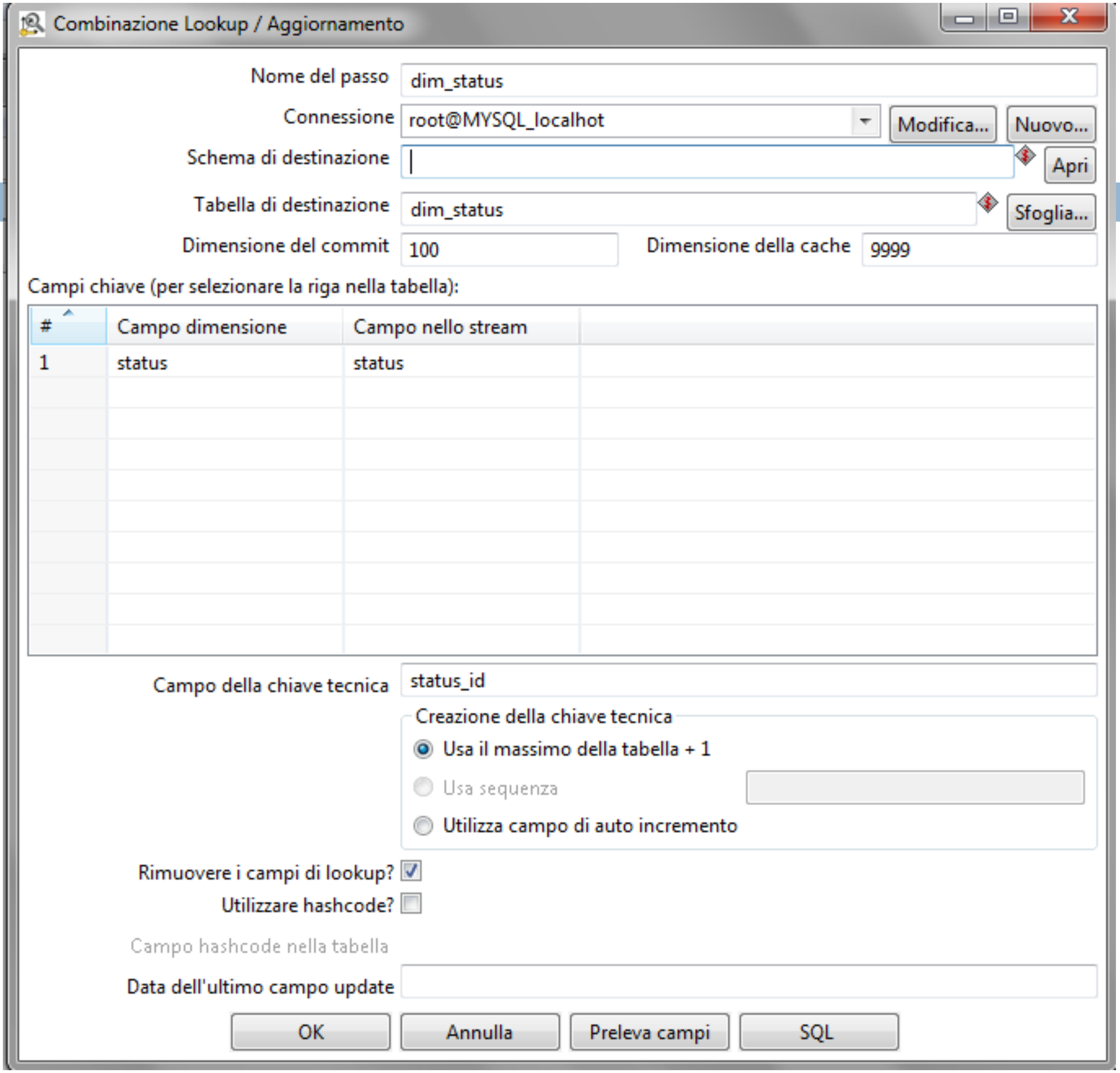
Creazione della chiave tecnica
- ● Usa il massimo della tabella + 1
- ○ Usa sequenza
- ○ Utilizza campo di auto incremento

Rimuovere i campi di lookup? ☑

Utilizzare hashcode? ☐

Campo hashcode nella tabella

Data dell'ultimo campo update


Combinazione Lookup / Aggiornamento

Nome del passo: dim_status

Connessione: root@MYSQL_localhot

Schema di destinazione:

Tabella di destinazione: dim_status

Dimensione del commit: 100

Dimensione della cache: 9999

Campi chiave (per selezionare la riga nella tabella):

#	Campo dimensione	Campo nello stream
1	status	status

Campo della chiave tecnica: status_id

Creazione della chiave tecnica
- ● Usa il massimo della tabella + 1
- ○ Usa sequenza
- ○ Utilizza campo di auto incremento

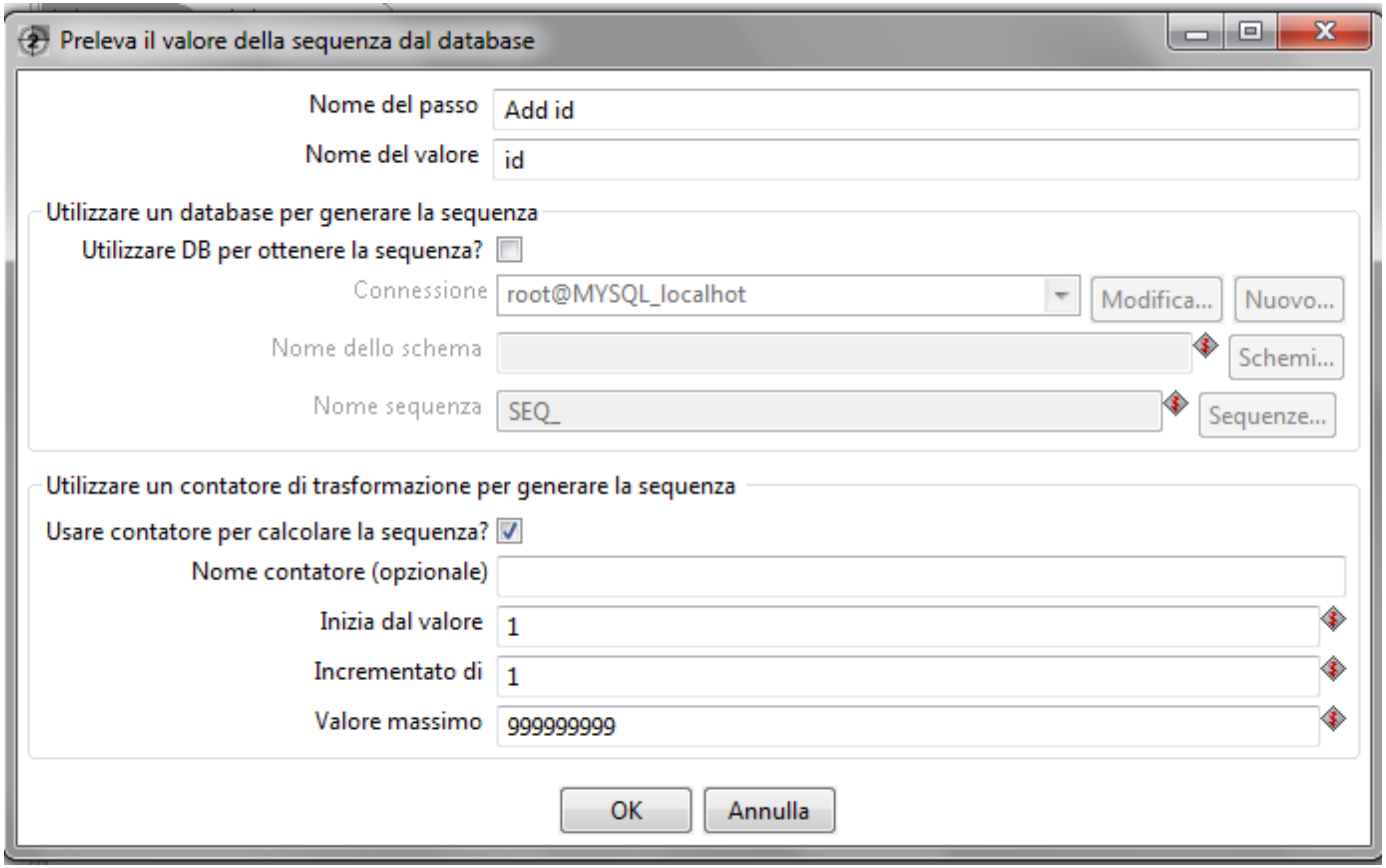
Rimuovere i campi di lookup? ✓

Utilizzare hashcode? ☐

Campo hashcode nella tabella

Data dell'ultimo campo update:

Add sequence



Preleva il valore della sequenza dal database

Nome del passo	Add id
Nome del valore	id

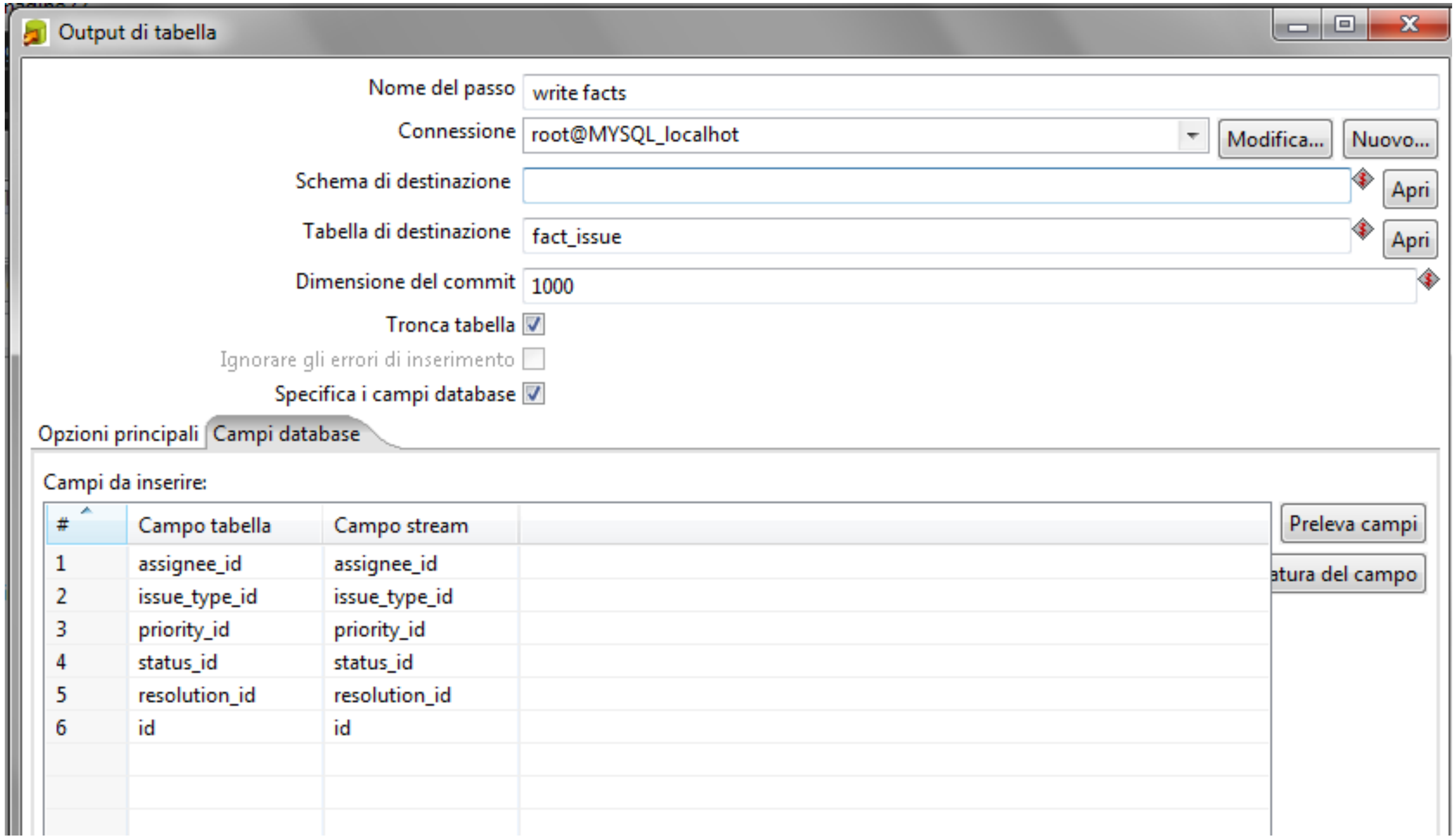
Utilizzare un database per generare la sequenza

Utilizzare DB per ottenere la sequenza?	☐
Connessione	root@MYSQL_localhot
Nome dello schema	
Nome sequenza	SEQ_

Utilizzare un contatore di trasformazione per generare la sequenza

Usare contatore per calcolare la sequenza?	✓
Nome contatore (opzionale)	
Inizia dal valore	1
Incrementato di	1
Valore massimo	999999999

Output table



Output di tabella

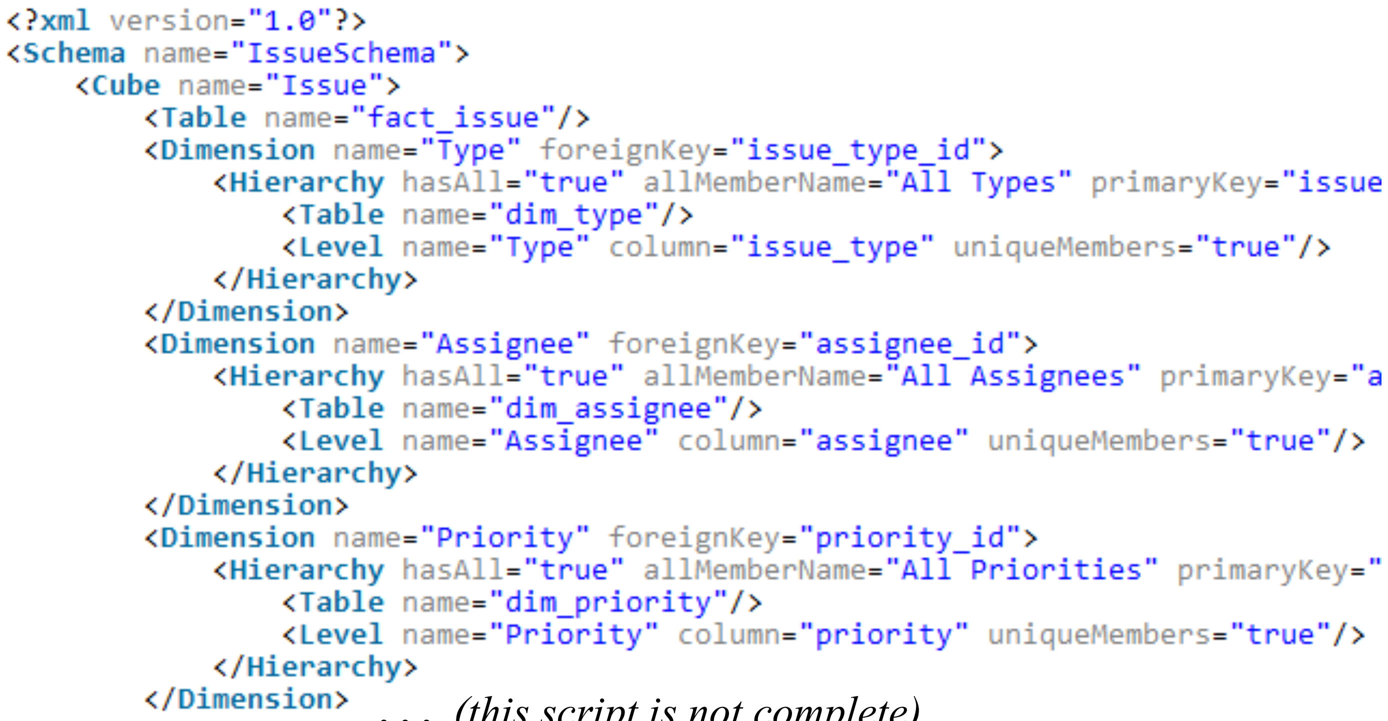
Nome del passo	write facts
Connessione	root@MYSQL_localhot
Schema di destinazione	
Tabella di destinazione	fact_issue
Dimensione del commit	1000
Tronca tabella	✓
Ignorare gli errori di inserimento	
Specifica i campi database	✓

Campi da inserire:

#	Campo tabella	Campo stream
1	assignee_id	assignee_id
2	issue_type_id	issue_type_id
3	priority_id	priority_id
4	status_id	status_id
5	resolution_id	resolution_id
6	id	id

The Schema File

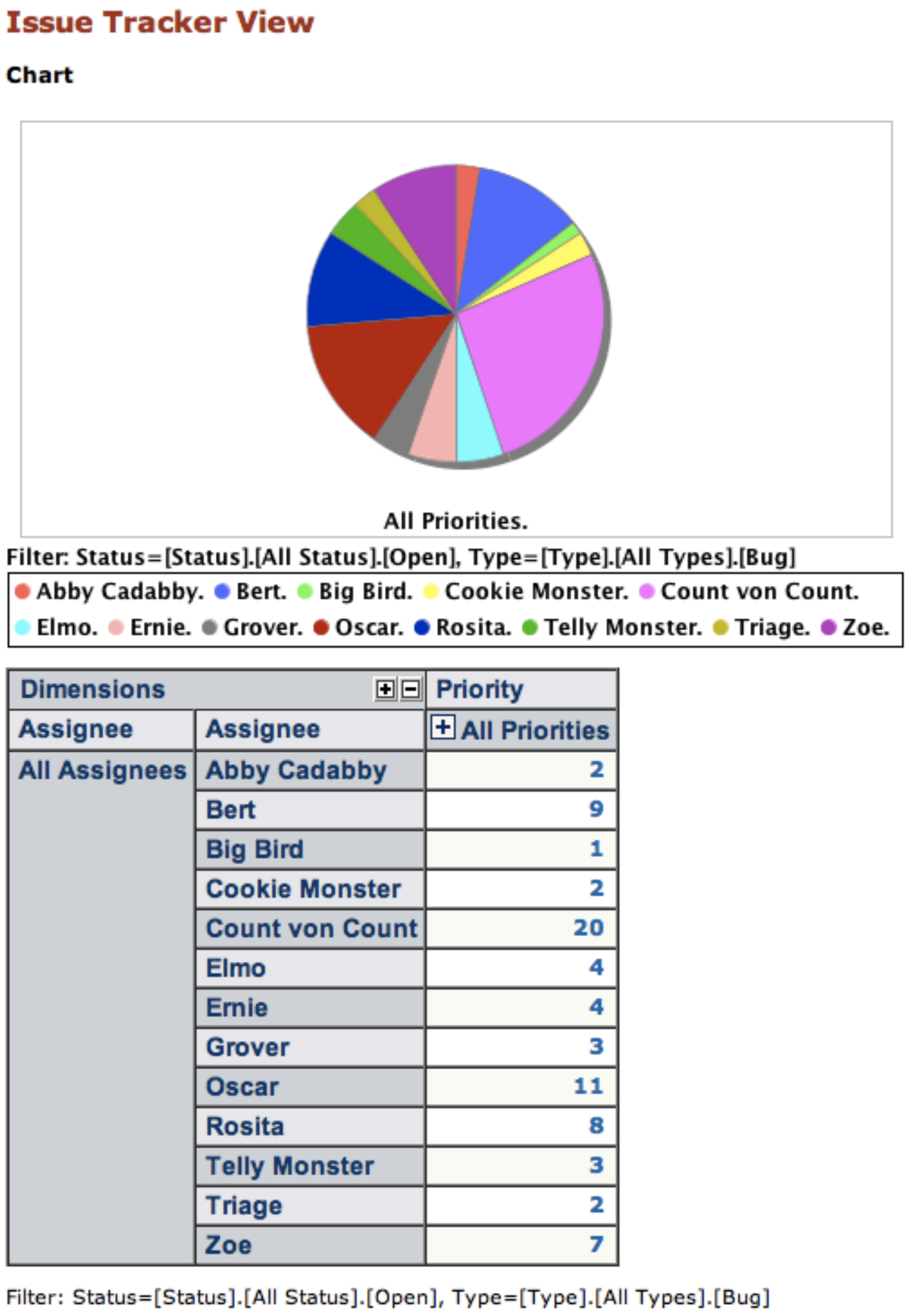
- After storing the cube in a DB, Mondrian must be informed about the table structure.
- The schema file now references a table for each dimension and specifies the key fields for both sides of each relation.



```
<?xml version="1.0"?>
<Schema name="IssueSchema">
    <Cube name="Issue">
        <Table name="fact_issue"/>
        <Dimension name="Type" foreignKey="issue_type_id">
            <Hierarchy hasAll="true" allMemberName="All Types" primaryKey="issue
                <Table name="dim_type"/>
                <Level name="Type" column="issue_type" uniqueMembers="true"/>
            </Hierarchy>
        </Dimension>
        <Dimension name="Assignee" foreignKey="assignee_id">
            <Hierarchy hasAll="true" allMemberName="All Assignees" primaryKey="a
                <Table name="dim_assignee"/>
                <Level name="Assignee" column="assignee" uniqueMembers="true"/>
            </Hierarchy>
        </Dimension>
        <Dimension name="Priority" foreignKey="priority_id">
            <Hierarchy hasAll="true" allMemberName="All Priorities" primaryKey="
                <Table name="dim_priority"/>
                <Level name="Priority" column="priority" uniqueMembers="true"/>
            </Hierarchy>
        </Dimension>
```

… *(this script is not complete)*

Using JPivot



Issue Tracker View

Chart

All Priorities.

Filter: Status=[Status].[All Status].[Open], Type=[Type].[All Types].[Bug]

● Abby Cadabby. ● Bert. ● Big Bird. ● Cookie Monster. ● Count von Count. ● Elmo. ● Ernie. ● Grover. ● Oscar. ● Rosita. ● Telly Monster. ● Triage. ● Zoe.

Dimensions		Priority
Assignee	Assignee	All Priorities
All Assignees	Abby Cadabby	2
	Bert	9
	Big Bird	1
	Cookie Monster	2
	Count von Count	20
	Elmo	4
	Ernie	4
	Grover	3
	Oscar	11
	Rosita	8
	Telly Monster	3
	Triage	2
	Zoe	7

Filter: Status=[Status].[All Status].[Open], Type=[Type].[All Types].[Bug]

Installation Tools

- You can download the tools from http://community.pentaho.com

1. **Pentaho BI Platform and Server**
   - Stable build of Pentaho BI Server 3.7.0 or higher

2. **Schema Workbench**
   - Stable build of Schema Workbench 3.2.1 or higher

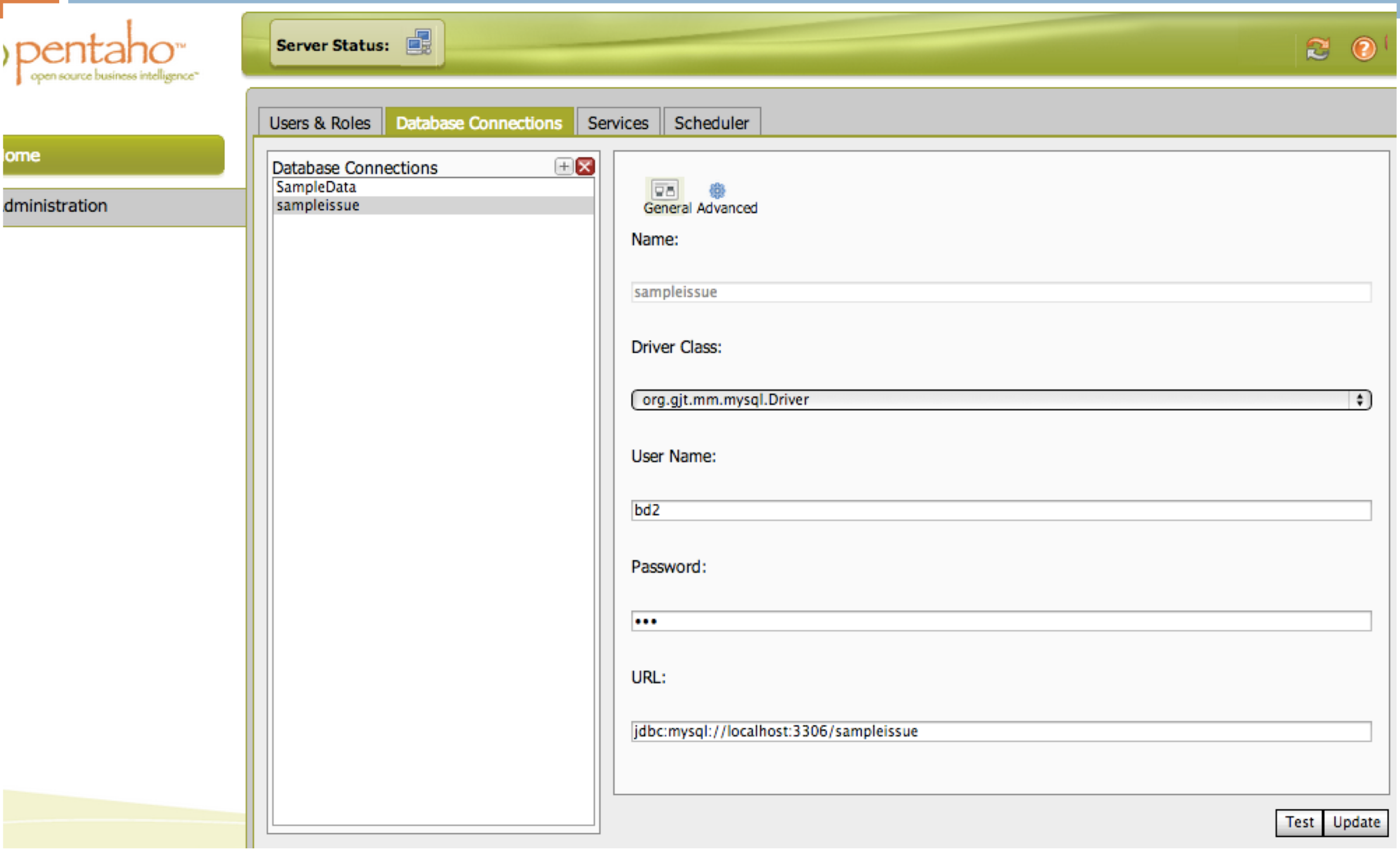
Pentaho BI Platform and Server

- **biserver-ce**
  - This is the actual Pentaho BI Server (Community Edition).
  - Set the variable **JAVA_HOME** to the JDK distribution (not to JRE).
  - *Start this server before the other tools.*
  - The server URL is: http://localhost:8080/pentaho

Pentaho BI Platform and Server

- **administration-console (PAC)**
  - This is an administrative service to manage and configure the actual Pentaho BI Server.
  - Pentaho Administration Console (PAC) URL: http://localhost:8099
  - Login: ***admin***, Password: ***password***
  - *Configure an user and a connection to the MySQL DBMS.*
    - The configured user and connection allows the access to the Mondrian server.

Database connection



SampleData
sampleissue

Name:

sampleissue

Driver Class:

org.gjt.mm.mysql.Driver

User Name:

bd2

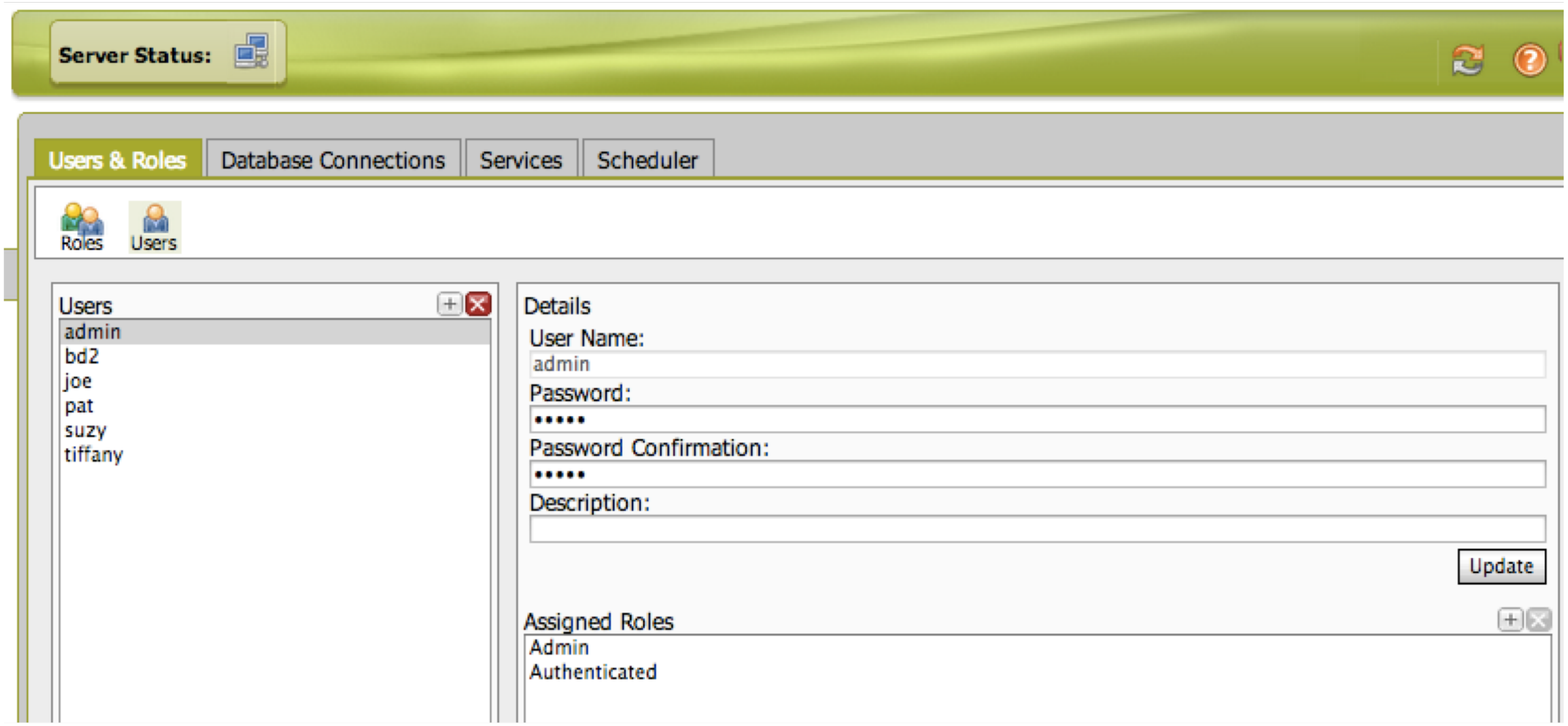
Password:

•••

URL:

jdbc:mysql://localhost:3306/sampleissue

Users and roles (access to the server)



Users
admin
bd2
joe
pat
suzy
tiffany

Details
User Name:
admin
Password:
•••••
Password Confirmation:
•••••
Description:

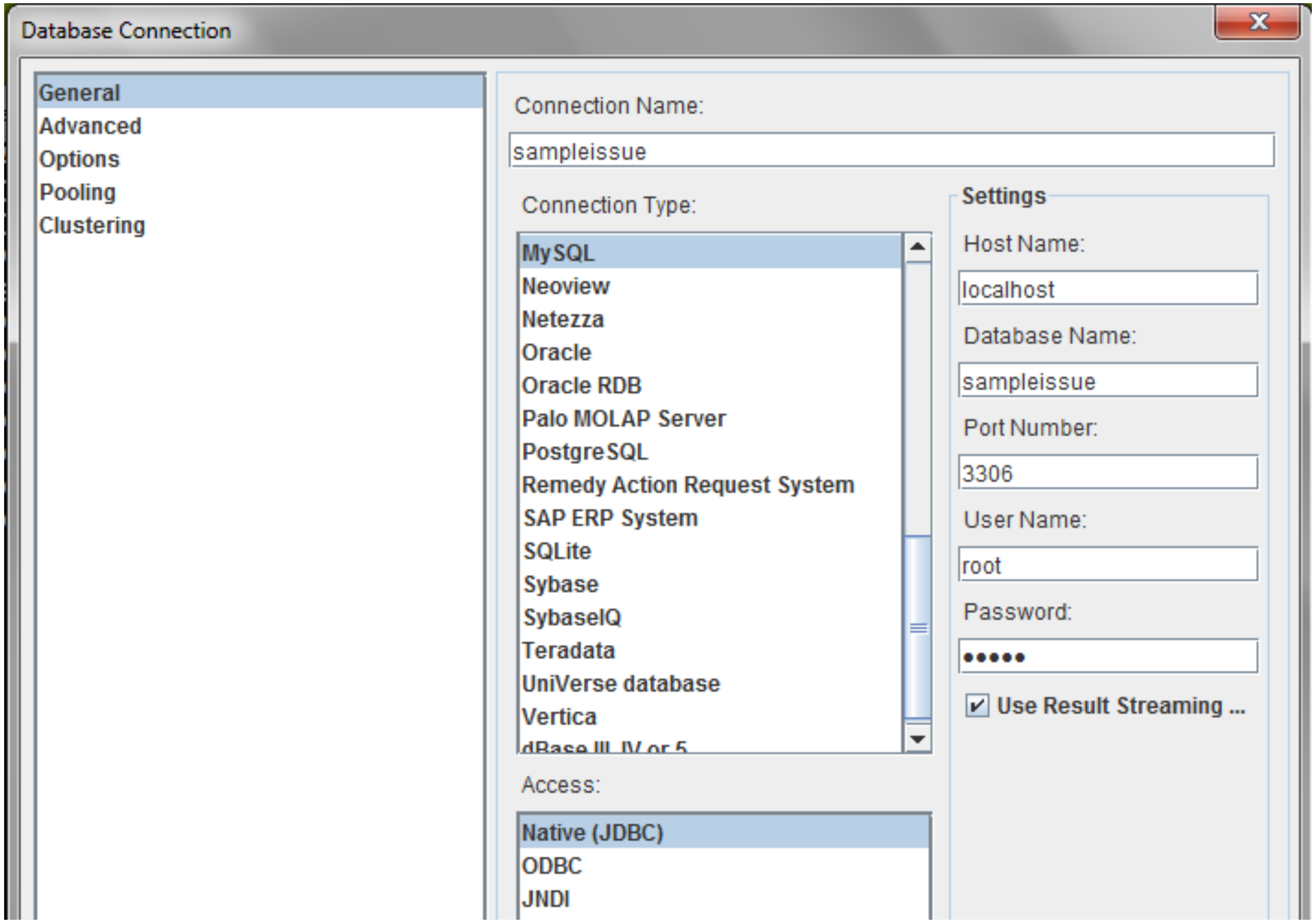
Assigned Roles
Admin
Authenticated

Pentaho Schema Workbench

- Pentaho Schema Workbench is distributed as .zip and .tar.gz archives.
- After downloading, you need to unpack the file.
- This yields a single directory called **schema-workbench** containing all the software.

- You need to place any JDBC Driver .jar files that you may need to connect to the data warehouse in the ***drivers*** directory:
  - Add the MySQL connector .jar file.

Database connection
(The MySQL DBMS must be active and accessible)


Database Connection

- General
- Advanced
- Options
- Pooling
- Clustering

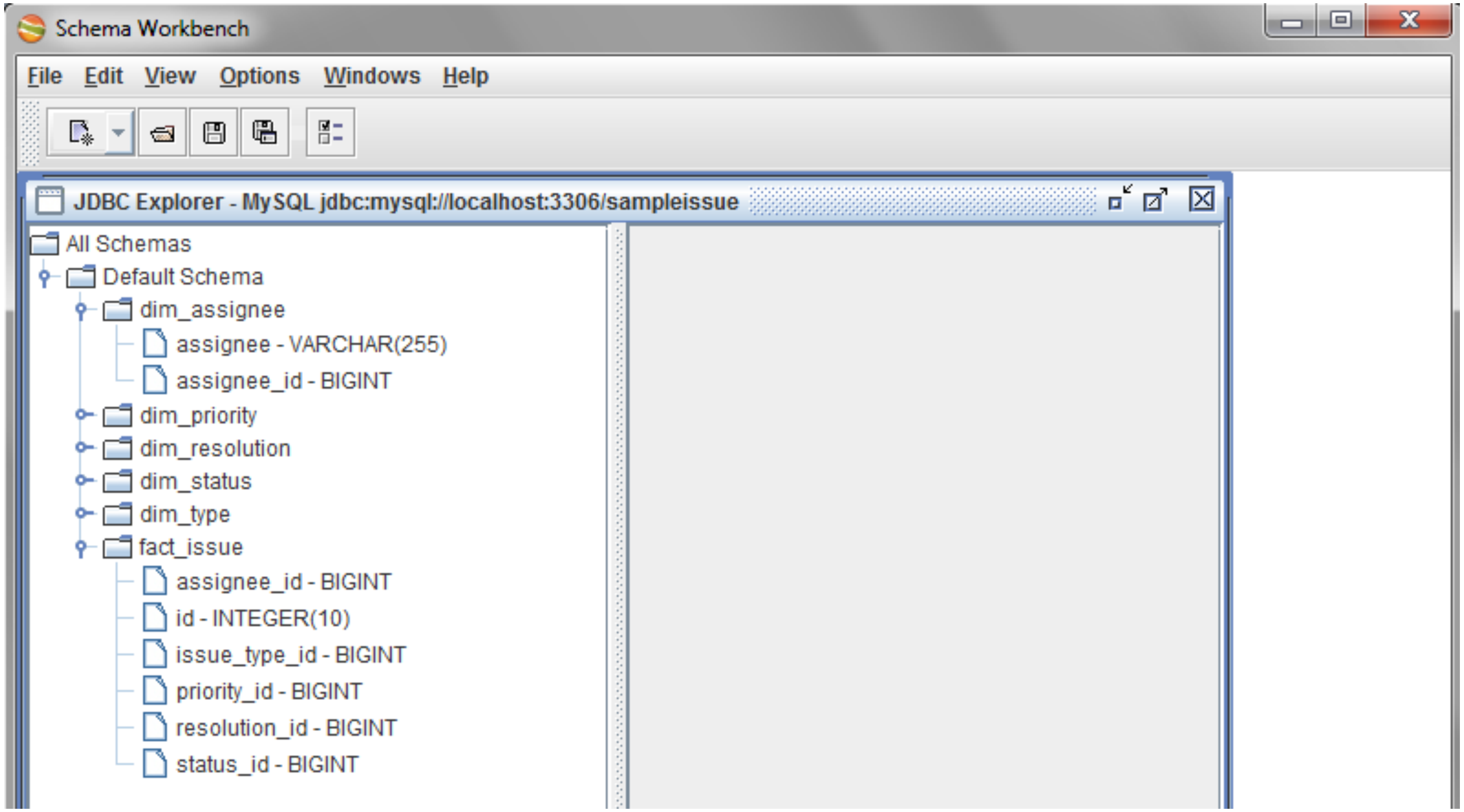
Connection Name: sampleissue

Connection Type:
- [✓ MySQL]
- Neoview
- Netezza
- Oracle
- Oracle RDB
- Palo MOLAP Server
- PostgreSQL
- Remedy Action Request System
- SAP ERP System
- SQLite
- Sybase
- SybaseIQ
- Teradata
- UniVerse database
- Vertica
- dBase III, IV or 5

Access:
- [✓ Native (JDBC)]
- ODBC
- JNDI

Settings
Host Name: localhost
Database Name: sampleissue
Port Number: 3306
User Name: root
Password: ●●●●●
✓ Use Result Streaming ...

JDBC Explorer



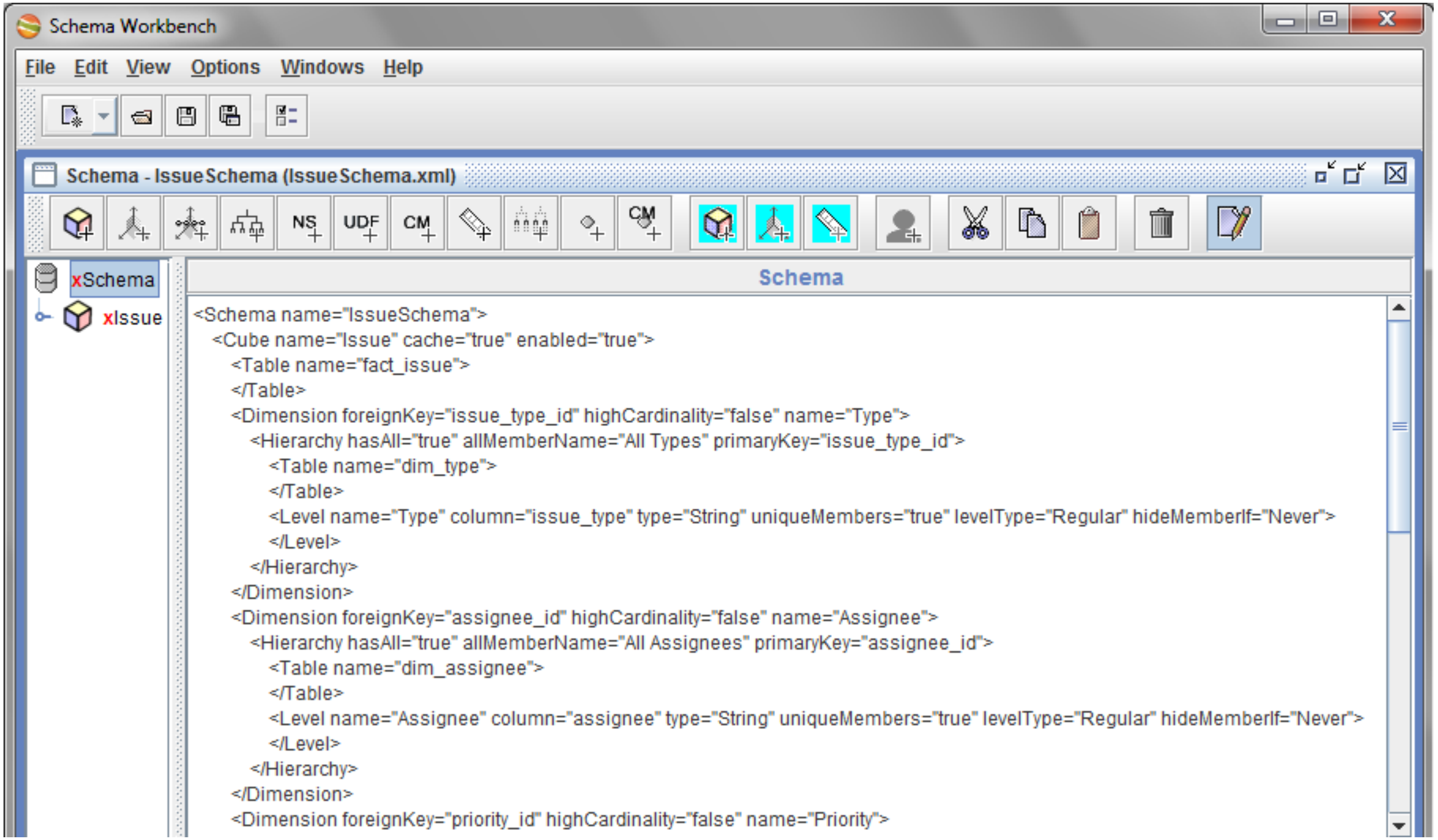
JDBC Explorer - MySQL jdbc:mysql://localhost:3306/sampleissue

```
All Schemas
  Default Schema
    dim_assignee
      assignee - VARCHAR(255)
      assignee_id - BIGINT
    dim_priority
    dim_resolution
    dim_status
    dim_type
    fact_issue
      assignee_id - BIGINT
      id - INTEGER(10)
      issue_type_id - BIGINT
      priority_id - BIGINT
      resolution_id - BIGINT
      status_id - BIGINT
```

Using the Schema Editor

- Schemas are created and edited using the schema editor.
- File → New → Schema to open the schema editor.
- The schema editor has a tree view on the left side, showing the contents of the schema.
  - Initially, this will be almost empty, save for the Schema node, which is the root of the entire schema.
- On the right side, the schema editor has a workspace where you can edit elements in the schema.

The XML document

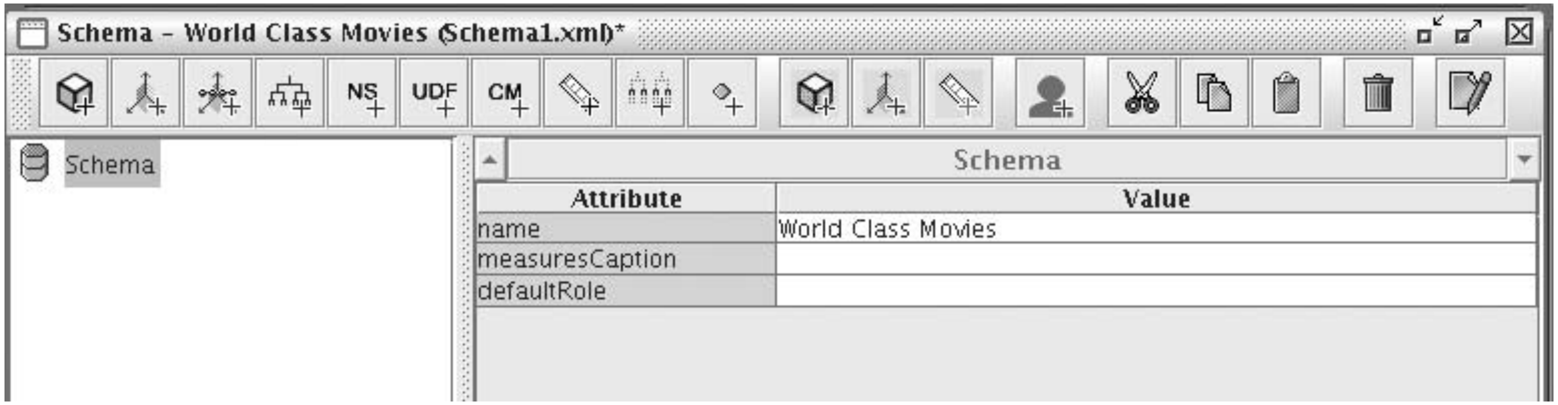

Schema Workbench
File Edit View Options Windows Help
Schema - IssueSchema (IssueSchema.xml)
NS UDF CM CM
xSchema
xIssue
Schema

```
<Schema name="IssueSchema">
  <Cube name="Issue" cache="true" enabled="true">
    <Table name="fact_issue">
    </Table>
    <Dimension foreignKey="issue_type_id" highCardinality="false" name="Type">
      <Hierarchy hasAll="true" allMemberName="All Types" primaryKey="issue_type_id">
        <Table name="dim_type">
        </Table>
        <Level name="Type" column="issue_type" type="String" uniqueMembers="true" levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension foreignKey="assignee_id" highCardinality="false" name="Assignee">
      <Hierarchy hasAll="true" allMemberName="All Assignees" primaryKey="assignee_id">
        <Table name="dim_assignee">
        </Table>
        <Level name="Assignee" column="assignee" type="String" uniqueMembers="true" levelType="Regular" hideMemberIf="Never">
        </Level>
      </Hierarchy>
    </Dimension>
    <Dimension foreignKey="priority_id" highCardinality="false" name="Priority">
```

Basic Schema Editing Tasks

- The tasks can be summarized as follows:
  - Creating a schema
  - Creating cubes
  - Choosing a fact table
  - Adding measures
  - Creating (shared) dimensions
  - Editing the default hierarchy and choosing a dimension table
  - Defining hierarchy levels
  - Optionally, adding more dimensions
  - Associating dimensions with cubes

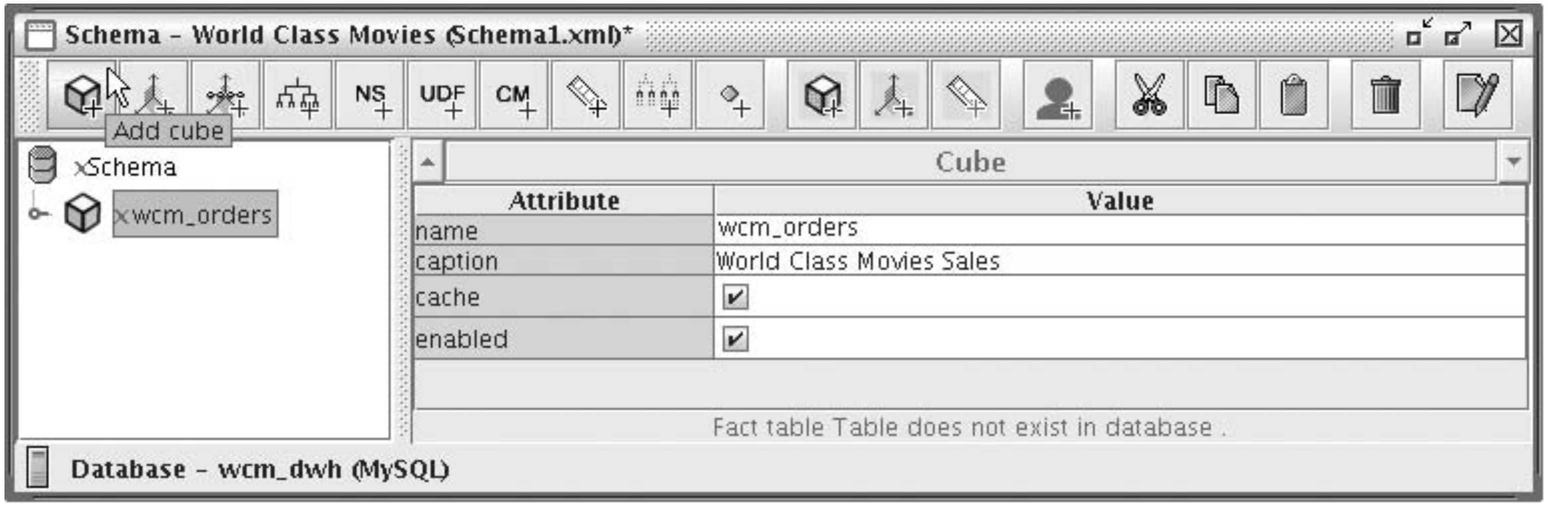
Creating a schema


Schema - World Class Movies (Schema1.xml)*

Attribute	Value
name	World Class Movies
measuresCaption	
defaultRole	

- In the Options menu: please, uncheck the "Require schema" option to avoid syntactical errors.

Creating a Cube


Schema - World Class Movies (Schema1.xml)*
Add cube
NS
UDF
CM
Schema
wcm_orders
Cube
Database - wcm_dwh (MySQL)

Attribute	Value
name	wcm_orders
caption	World Class Movies Sales
cache	✓
enabled	✓

Fact table Table does not exist in database .

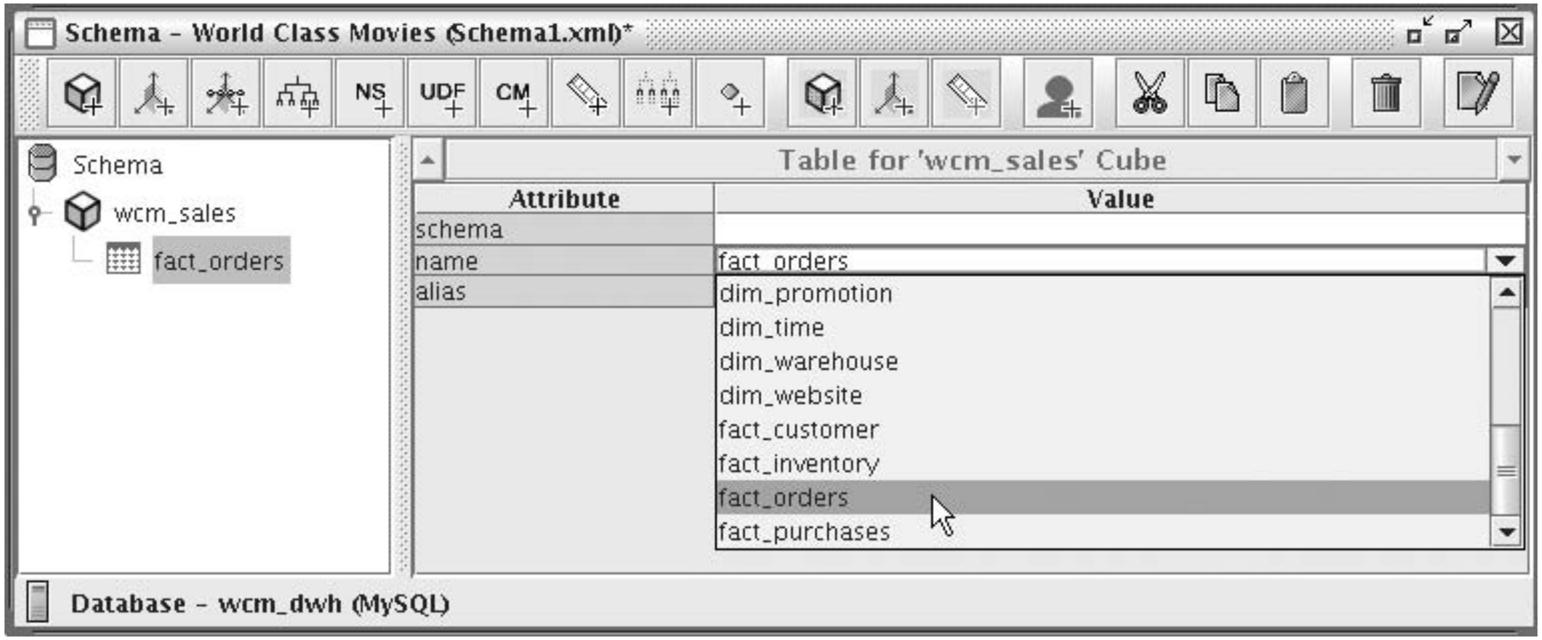
- **name** - Specifies the name that will be used in MDX queries to refer to this cube. This name must be unique within the schema.
- **caption** - Specifies a display name, which will be used by the user interface to present this cube to the end user.
- **cache** - Controls whether data from the fact table should be cached.
- **enabled** - Controls whether Mondrian should load or ignore the cube.

Errors

- A little red X icon can appear to the left of the schema and cube icons.
  - The red X icon indicates that there is some error or misconfiguration at or somewhere beneath that particular node.

Choosing a Fact Table

- The cube node is initially collapsed, and if you expand it, you will notice it contains a table node.
- This table node represents the fact table upon which the cube is built. In this case, your cube should be based on the **fact_orders** table,
  - which is why you set the table name using the drop-down list box.



Schema - World Class Movies (Schema1.xml)*

Schema
wcm_sales
fact_orders

Table for 'wcm_sales' Cube

Attribute	Value
schema	
name	fact_orders
alias	

dim_promotion
dim_time
dim_warehouse
dim_website
fact_customer
fact_inventory
fact_orders
fact_purchases

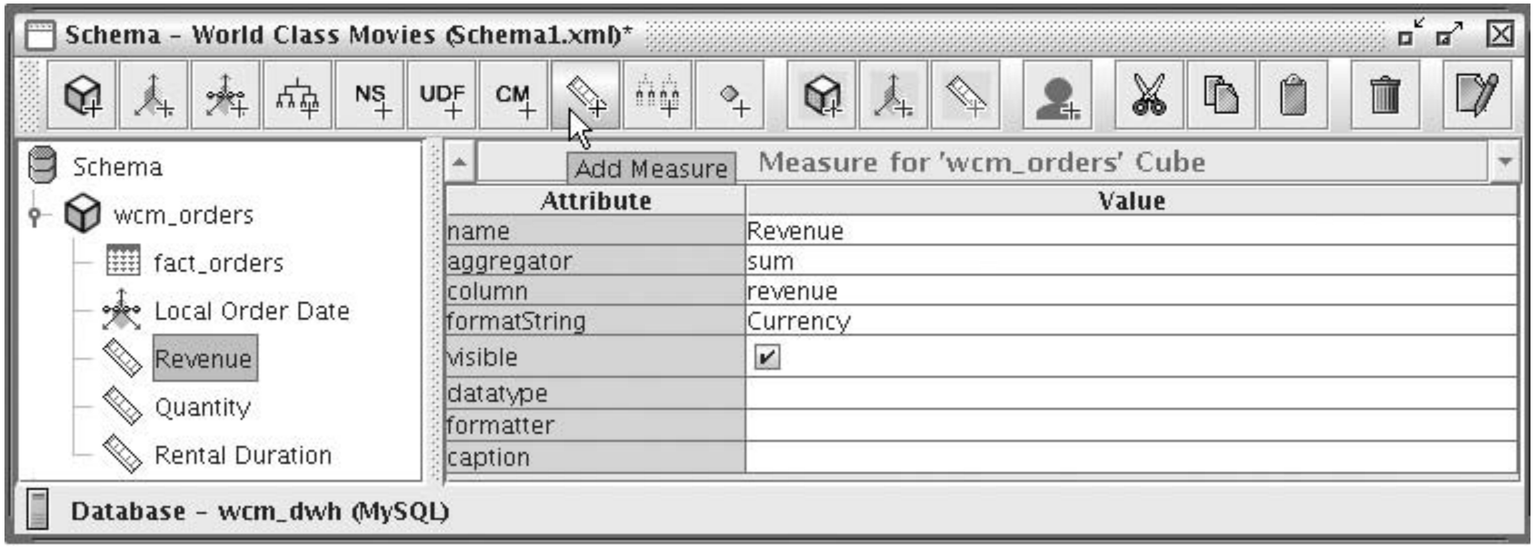
Database - wcm_dwh (MySQL)

Choosing a Fact Table (2)

- **schema** -The identifier of the database schema that contains the fact table.
  - When not explicitly specified, the default schema of the database connection is used.
- **name** - The name of the fact table.
  - When connected to a database, the property editor provides a drop-down list box.
- **alias** - This is the table alias that will be used for this table when generating SQL statements.
  - It may be useful to specify this in case you want to debug the SQL statements generated by Mondrian.

Adding Measures

- To add measures, first select the cube (or its fact table) in the tree view.
- Then, click the Add Measure button on the toolbar.
  - The order in which you specify the measures is significant: implicitly, the first measure in the cube is considered the default measure.



Measure for 'wcm_orders' Cube

Attribute	Value
name	Revenue
aggregator	sum
column	revenue
formatString	Currency
visible	✓
datatype	
formatter	
caption	

Adding Measures (2)

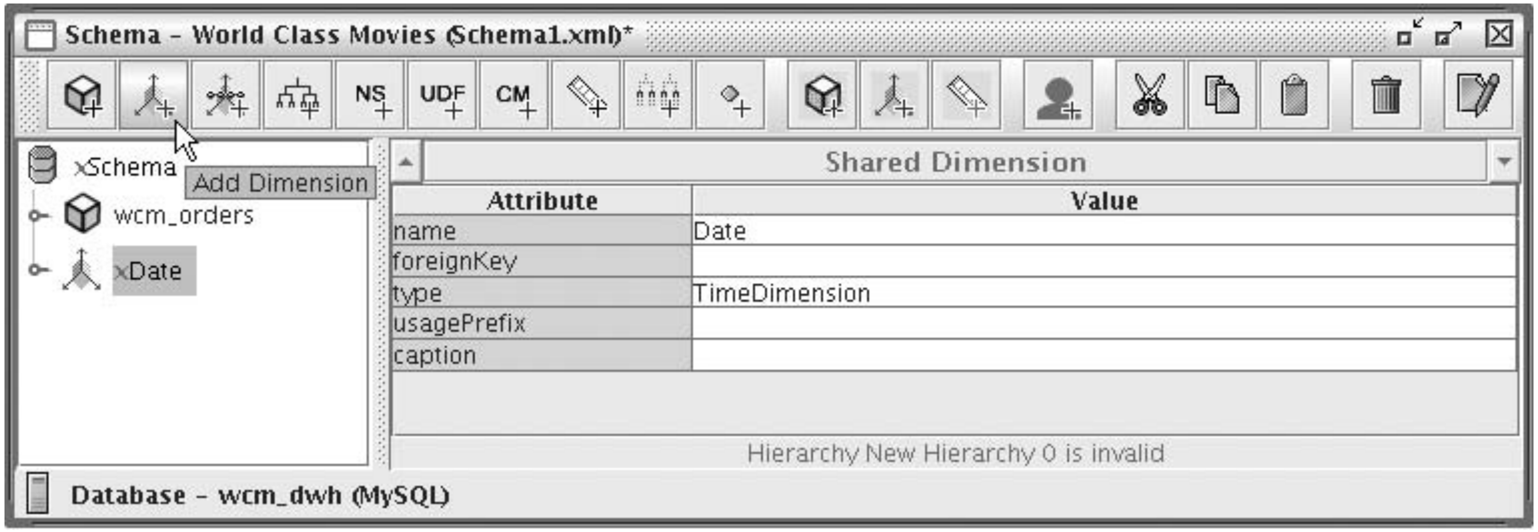
- **name** - The identifier that will be used to refer to this measure in MDX queries. *This must be unique within the cube*.
- **aggregator** - The name of the function that is used to aggregate the measure. The attribute grid offers a drop-down list box from where you can pick one of *sum*, *count*, *min*, *max*, *avg*, and *distinct-count*.
- **column** - The name of a column from the cube's fact table. When connected to the database, the attribute editor offers a drop-down list box from which you can pick the column.
- **formatString** - Here you can specify a string pattern that specifies how the measure value will be formatted for display.

Adding Measures (3)

- **visible** - A flag that specifies whether the measure is displayed to the end user in the user interface.
- **datatype** - Here you can use a drop-down list box to choose *String*, *Numeric*, *Integer*, *Boolean*, *Date*, *Time*, or *Timestamp*.
  - When returning data, the specified data type will be used to return data in the MDX result.
- **formatter** - You can use this attribute to specify a custom cell formatter.
- **caption** - Specifies the display name that is used to present this measure in the user interface.

Adding Dimensions

- The Mondrian schemas can contain dimensions in two places:
  - **Inside the cube that "owns" the dimension**
    - These dimensions are called private dimensions because they are known only to the cube that contains it and cannot be used outside the enclosing cube.
  - **Inside the schema itself**
    - These are *shared dimensions and can be* associated with multiple cubes, and/or multiple times with the same cube.


Schema - World Class Movies (Schema1.xml)*
NS
UDF
CM
xSchema
Add Dimension
wcm_orders
xDate
Shared Dimension
Database - wcm_dwh (MySQL)

Attribute	Value
name	Date
foreignKey	
type	TimeDimension
usagePrefix	
caption	

Hierarchy New Hierarchy 0 is invalid

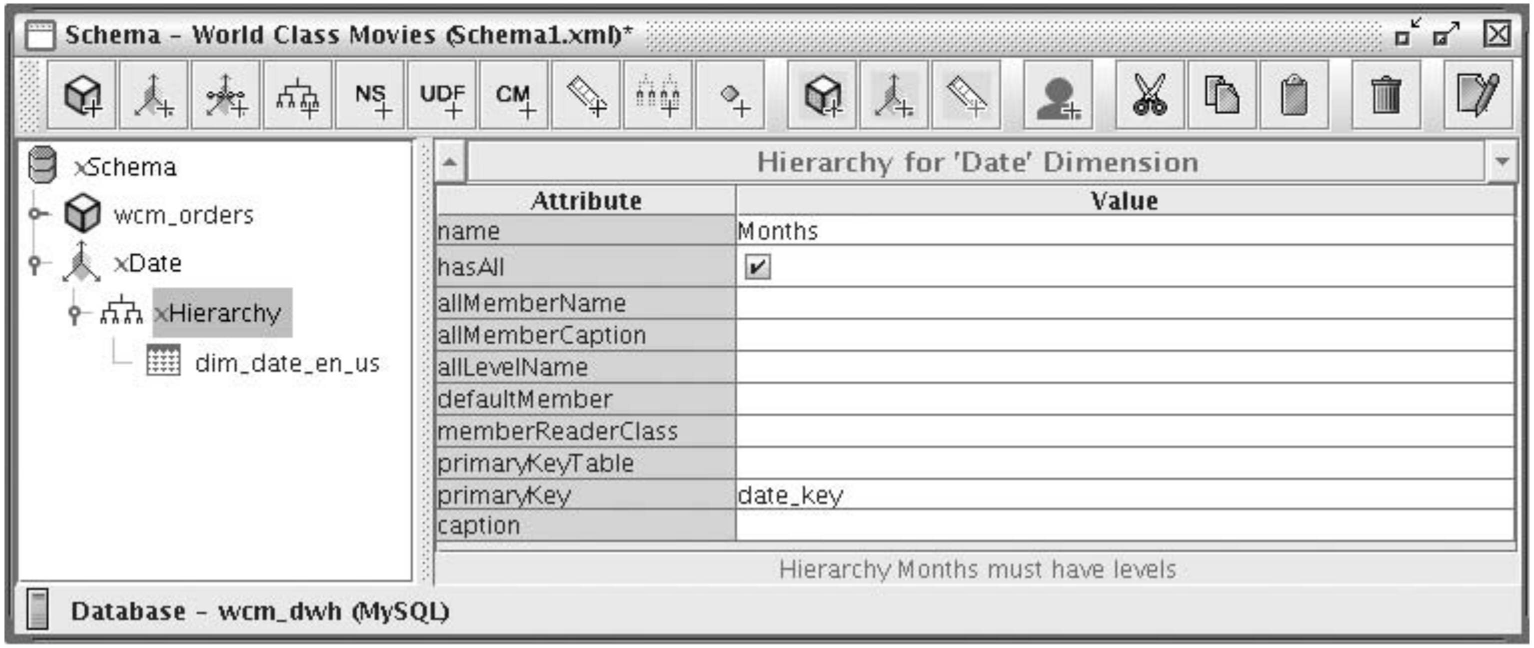
Adding Dimensions (2)

- **name** -
  - For private dimensions, the name refers to this dimension in MDX queries.
    - The name must be unique among all other dimensions used by the cube.
  - For shared dimensions, the name refers to the dimension when you are associating it with a cube.
    - The name must be unique within the schema.
- **foreignKey** - If this is a private dimension, this is the name of a column from the cube's fact table that refers to the dimension table that corresponds to this dimension.
- **type** - If your dimension is time or date related, you should use ***TimeDimension***. This allows you to use the standard MDX time and date functions. Otherwise, use ***StandardDimension***.
- **caption** - This is a display name used to present this dimension to the end user via the user interface.

Adding and Editing Hierarchies and Choosing Dimension Tables

- When you create a dimension, a new hierarchy is also created.
  - You can see it when you expand the dimension node.
- In addition, a table node must be created beneath the hierarchy node.
  - Before you edit the hierarchy node, it is best to configure the underlying table node.
  - The table node represents the dimension table that will deliver the values for the levels of the hierarchy.
  - The procedure to configure the table is exactly the same as the procedure for choosing a fact table for a cube, which was described earlier in this section.

Adding and Editing Hierarchies and Choosing Dimension Tables (2)


Schema - World Class Movies (Schema1.xml)*

Hierarchy for 'Date' Dimension

Attribute	Value
name	Months
hasAll	✓
allMemberName	
allMemberCaption	
allLevelName	
defaultMember	
memberReaderClass	
primaryKeyTable	
primaryKey	date_key
caption	

Hierarchy Months must have levels

Database - wcm_dwh (MySQL)

- **name** - The name used in MDX queries to refer to the hierarchy.
  - It must be unique within the dimension.
- **caption** - The name that is used to present this hierarchy to the end user in the user interface.

Adding and Editing Hierarchies and Choosing Dimension Tables (3)

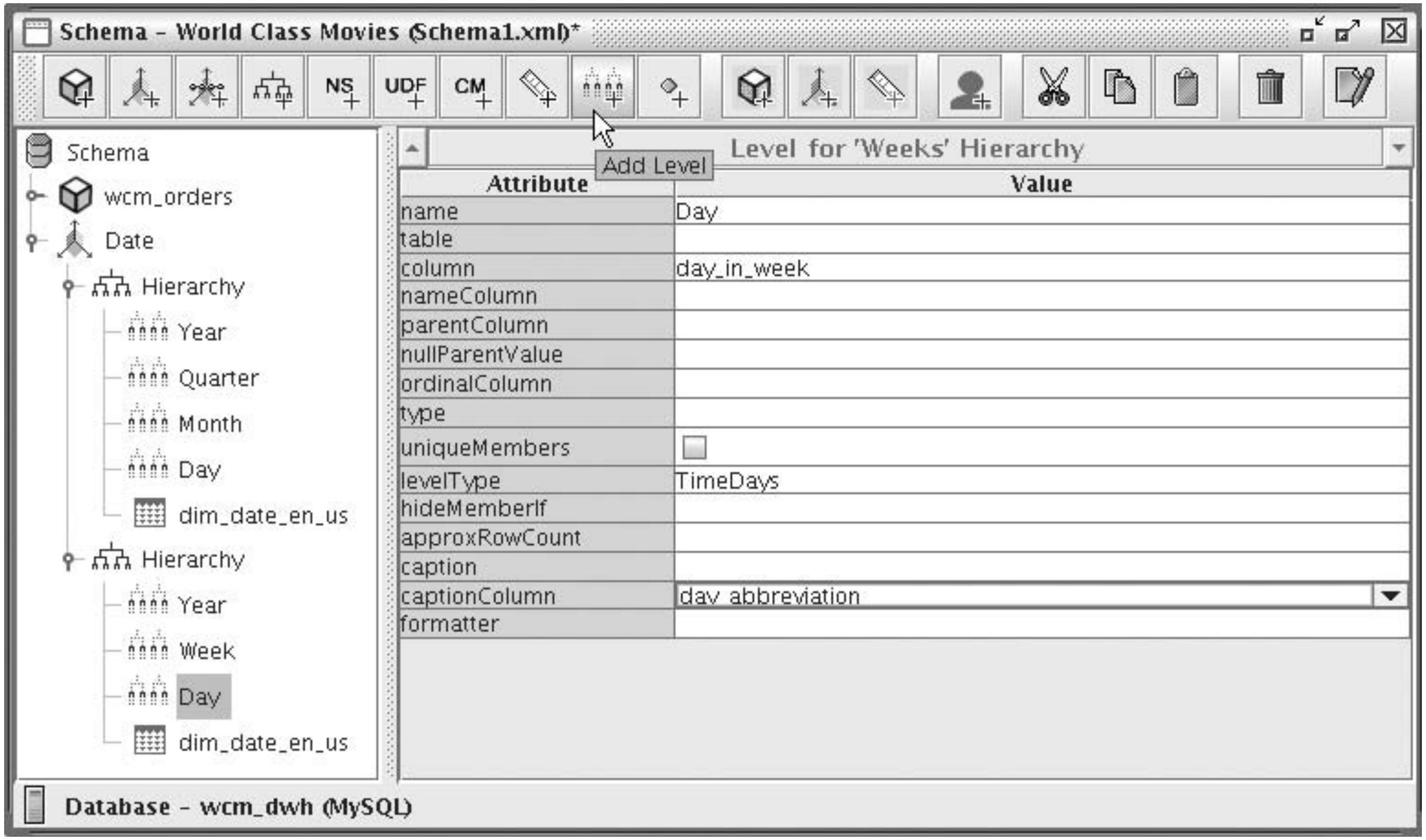
- **hasAll** - A flag that indicates whether the hierarchy should have an all level with an all member.
  - **Es:** a single member in the top of the hierarchy that represents all other members. Usually you should leave this on.
- **allMemberName** - If hasAll is enabled, this specifies the MDX identifier that is to be used for the all member.
- **allMemberCaption** - If hasAll is enabled, you can use this to specify the name that will be used to present the all member to the end user in the user interface.
- **allLevelName** - The name used to refer to the all level in MDX queries.
- **defaultMember** - The name of the default member. If this is not specified, then the all member will be used as default member if the hierarchy has an All member.

Adding and Editing Hierarchies and Choosing Dimension Tables (4)

- **primaryKey** - Typically, you should use this to specify the name of the primary key column of this hierarchy's dimension table.
  - *To be exact:* this is the column name of the dimension table that is referenced by the rows in the fact table. This should be a column in this hierarchy's dimension table.

Adding Hierarchy Levels

- Now that you created the hierarchies, you must define their levels.



Level for 'Weeks' Hierarchy

Attribute	Value
name	Day
table	
column	day_in_week
nameColumn	
parentColumn	
nullParentValue	
ordinalColumn	
type	
uniqueMembers	☐
levelType	TimeDays
hideMemberIf	
approxRowCount	
caption	
captionColumn	day_abbreviation
formatter	

Adding Hierarchy Levels (2)

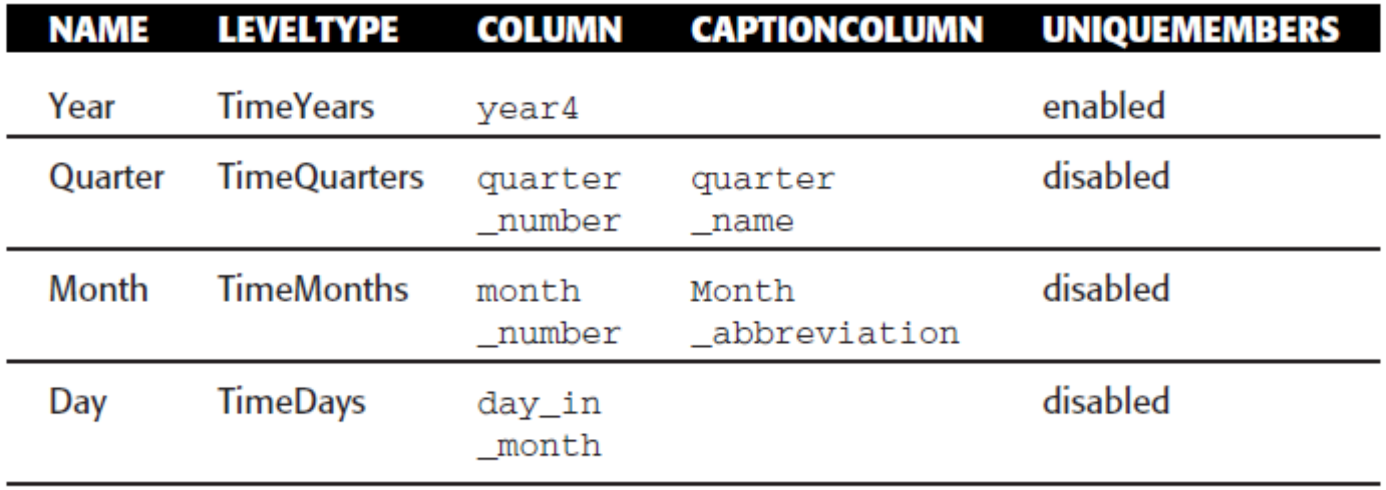
- **name** - The name that is used to refer to this level in MDX queries.
- **table** - The name of the table that contains the columns where the dimension data is stored for this level.
  - When not specified, the hierarchy's dimension table will be used. This is the normal situation for *star schemas* like the one used in this example.
  - You need to specify a particular table only when dealing with *snowflake schemas*.
- **column** - The column that represents the member identifier for this level. This must correspond to this level's table *(see the table attribute)*.
- **nameColumn** - The name of the column that contains the name of this level.
  - When not specified, the value of the name property is used. Typically you should leave this blank.
- **captionColumn** - You can specify which column of the level's dimension table should be used to present the members to the end user.
  - When not specified, the member identifier will be used.

Adding Hierarchy Levels (3)

- **ordinalColumn** - This attribute can be used to specify which column defines how the member values should be sorted by default.
- **type** - The data type of the member values. This is used to control if and how values must be quoted when generating SQL from MDX queries.
- **uniqueMembers** - A flag indicating whether all the members at this level have unique values.
    - This is always true for the first level (not counting the all level) of any hierarchy.
- **levelType** - If you leave this blank, it will be assumed this is a regular level, which is the correct value for most dimensions.
    - Dimensions that were configured to be of the type TimeDimension must specify one of the predefined types for TimeDimension levels: TimeYears, TimeQuarters, TimeMonths, TimeWeeks, and TimeDays.
    - For TimeDimensions, specifying the levelType is a prerequisite for correct usage of the Mondrian date/time functions such as YTD.
- **hideMemberIf** - This determines in which cases a member should be hidden. Typically, you can leave this blank, which is equivalent to setting the value to Never. In this case, the member is always shown.

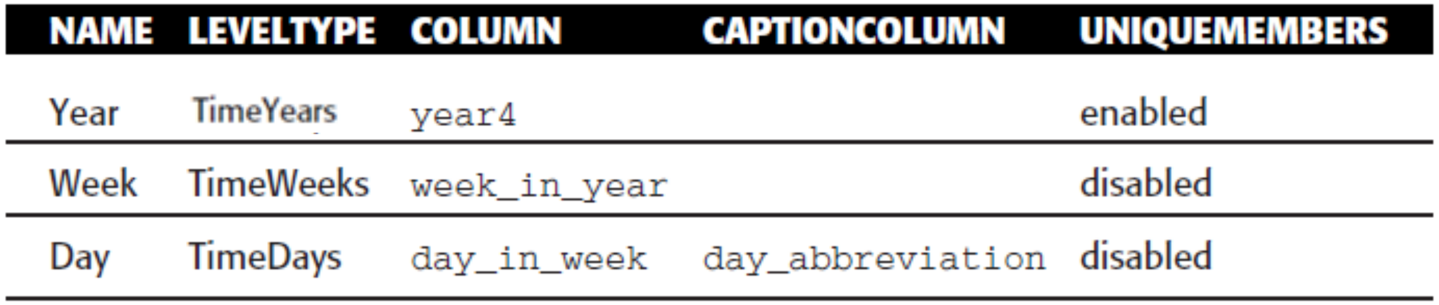
Example

- The levels of the Months hierarchy



NAME	LEVELTYPE	COLUMN	CAPTIONCOLUMN	UNIQUEMEMBERS
Year	TimeYears	year4		enabled
Quarter	TimeQuarters	quarter _number	quarter _name	disabled
Month	TimeMonths	month _number	Month _abbreviation	disabled
Day	TimeDays	day_in _month		disabled

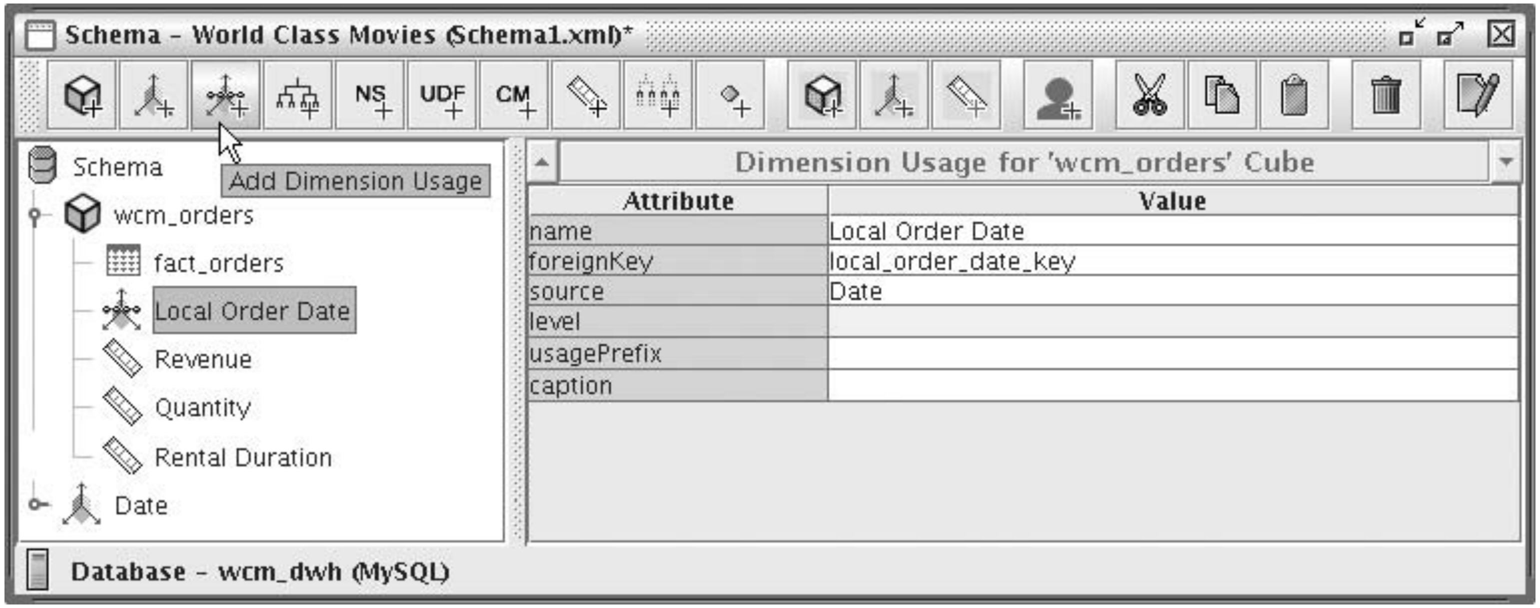
- The levels of the Weeks hierarchy



NAME	LEVELTYPE	COLUMN	CAPTIONCOLUMN	UNIQUEMEMBERS
Year	TimeYears	year4		enabled
Week	TimeWeeks	week_in_year		disabled
Day	TimeDays	day_in_week	day_abbreviation	disabled

Associating Cubes with Shared Dimensions

- In Mondrian schemas, the association between a cube and a shared dimension is called a *dimension usage.*
- *To add a dimension usage, either select the cube* and right-click the cube and choose the Add Dimension Usage option from the context menu.



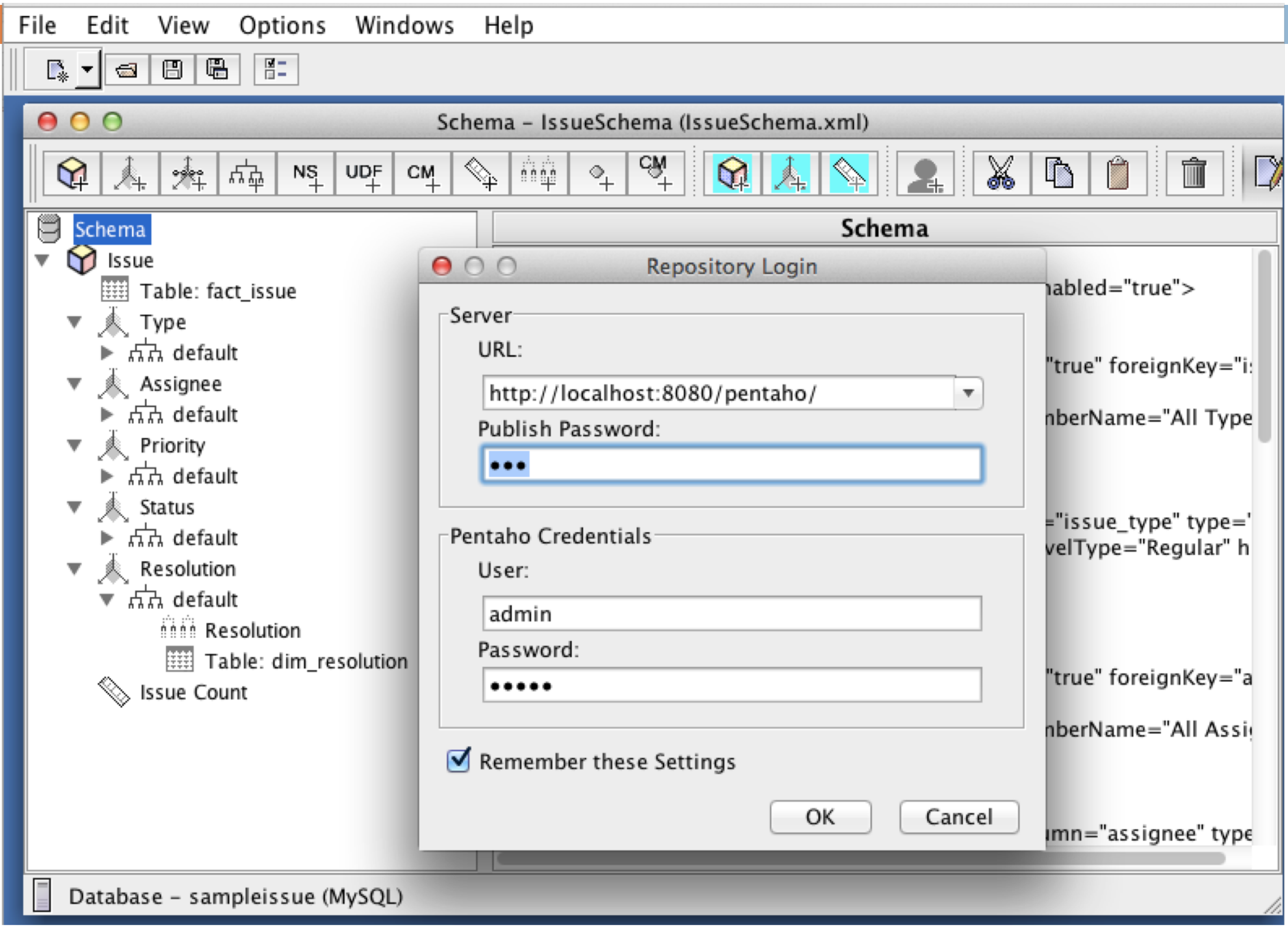
Dimension Usage for 'wcm_orders' Cube

Attribute	Value
name	Local Order Date
foreignKey	local_order_date_key
source	Date
level	
usagePrefix	
caption	

Publishing the Cube

- You can publish the cube to the Pentaho BI Server.
- To invoke the publish dialog, choose File → Publish from the main menu, and the dialog pops up.
- For the URL, specify the web address of the Pentaho BI Server to which you want to publish the schema.
  - You must use the publisher password that you specified in the server's ***publisher_config.xml*** file.
  - For the username and password, specify the credentials of the user created or modified with the **administration-console** tool.
- If the connection succeeds, a dialog appears that allows you to browse the server's solution repository.
  - Choose the appropriate path (or create a new folder).

Publishing the Cube



Server

URL: http://localhost:8080/pentaho/

Publish Password: •••

Pentaho Credentials

User: admin

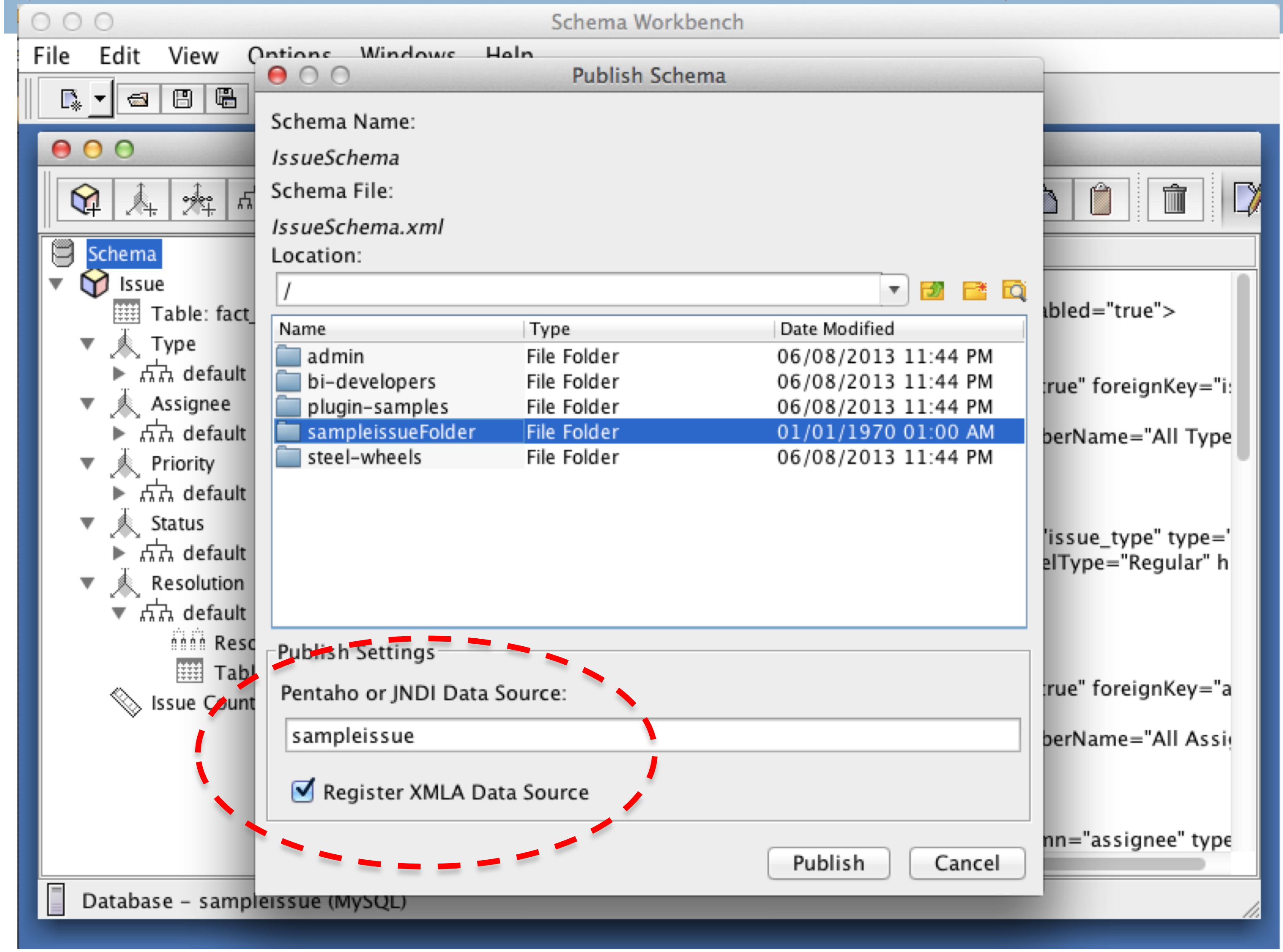
Password: •••••

[✓] Remember these Settings

abled="true">
"true" foreignKey="i
nberName="All Type
="issue_type" type='
velType="Regular" h
"true" foreignKey="a
nberName="All Assi
umn="assignee" type

Database - sampleissue (MySQL)

Publish the schema (use the database connection)



Schema Name:

IssueSchema

Schema File:

IssueSchema.xml

Location:

/

Name	Type	Date Modified
admin	File Folder	06/08/2013 11:44 PM
bi-developers	File Folder	06/08/2013 11:44 PM
plugin-samples	File Folder	06/08/2013 11:44 PM
sampleissueFolder	File Folder	01/01/1970 01:00 AM
steel-wheels	File Folder	06/08/2013 11:44 PM

Publish Settings

Pentaho or JNDI Data Source:

sampleissue

☑ Register XMLA Data Source

*Example: MDX Query Syntax**

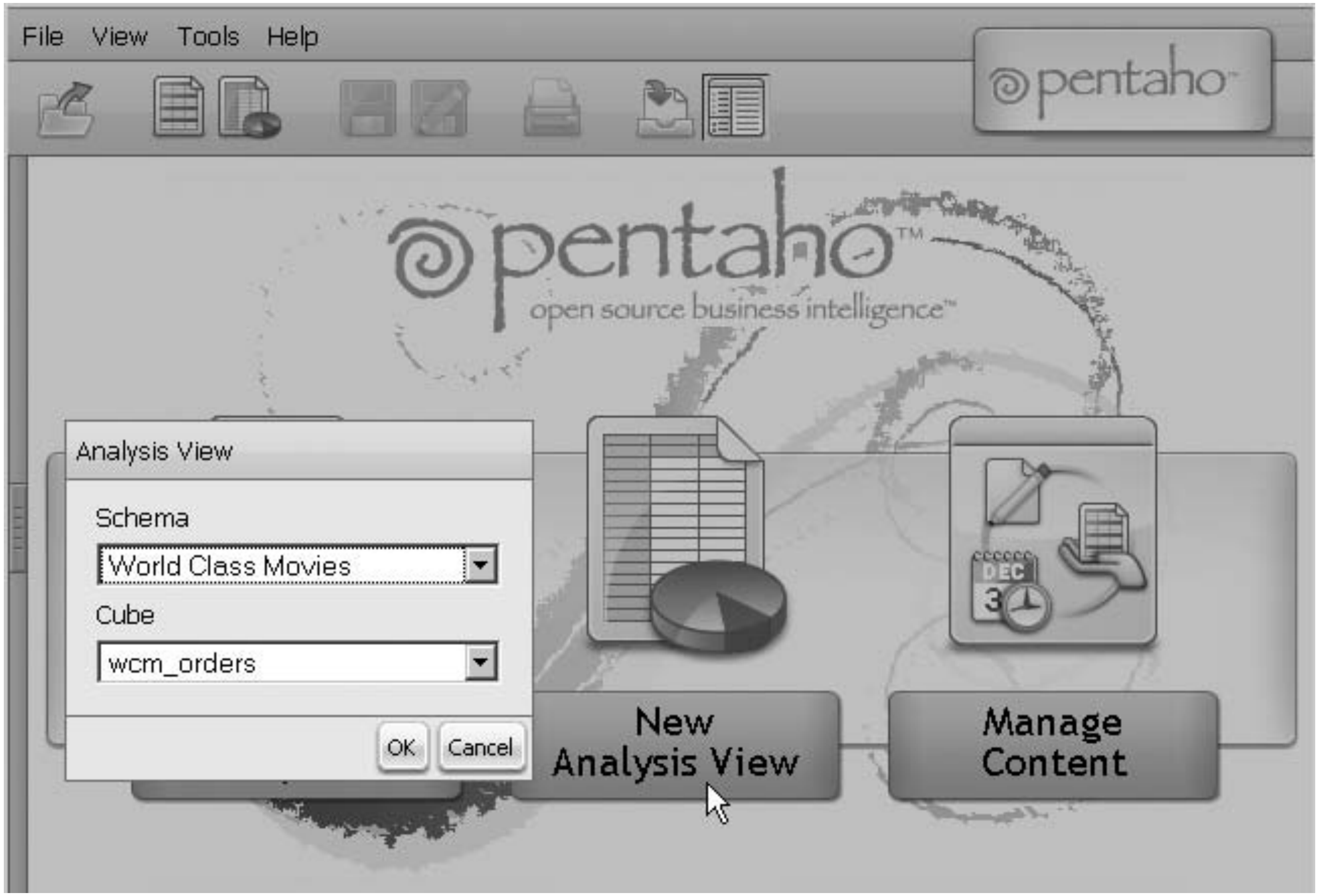
SELECT <member collection> **ON COLUMNS**,
<member collection> **ON ROWS**
FROM <cubename>
WHERE <conditions>

```
SELECT    { [Measures].[Store Sales] } ON COLUMNS,
          { [Date].[2002], [Date].[2003] } ON ROWS
FROM Sales
WHERE ( [Store].[USA].[CA] )
```

Visualizing Mondrian Cubes with JPivot

- The user console (http://localhost:8080/pentaho) of the Pentaho BI Server offers the possibility to create an *analysis view, which is essentially a JPivot cross table on top of a Mondrian* cube, wrapped in a Pentaho process action.
- To create a new analysis view, click the **analysis view** icon on the toolbar or on the initial workspace page.

Creating an analysis view

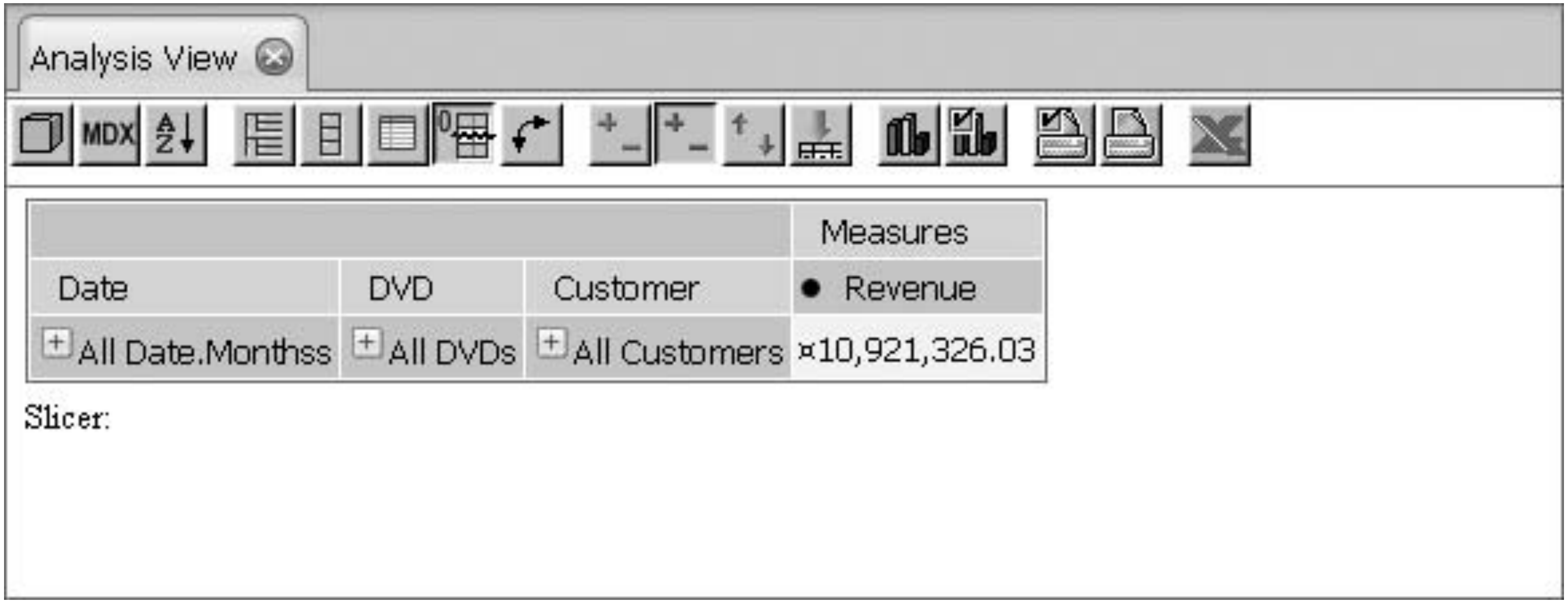


Analysis View

Schema: World Class Movies

Cube: wcm_orders

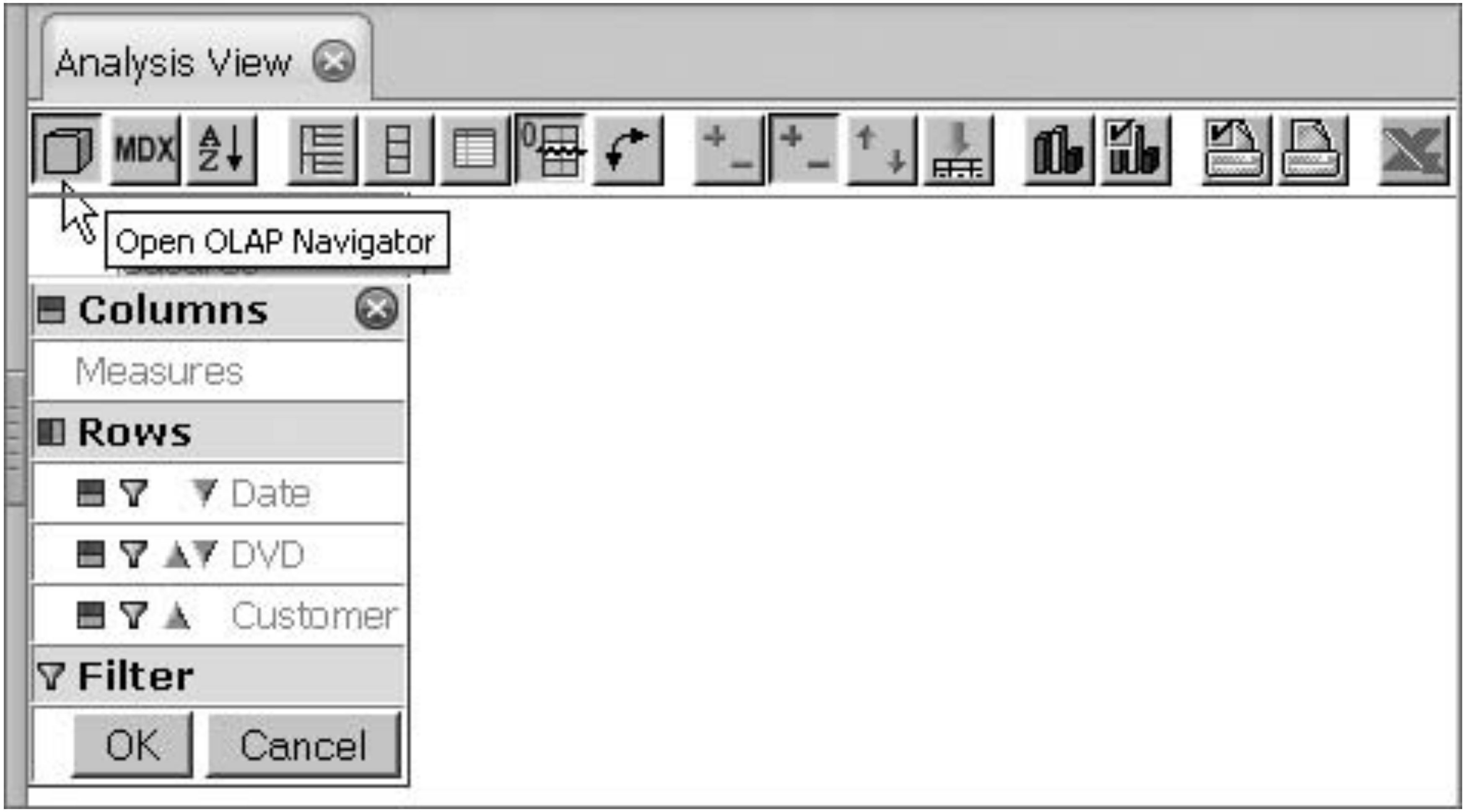
The default pivot table



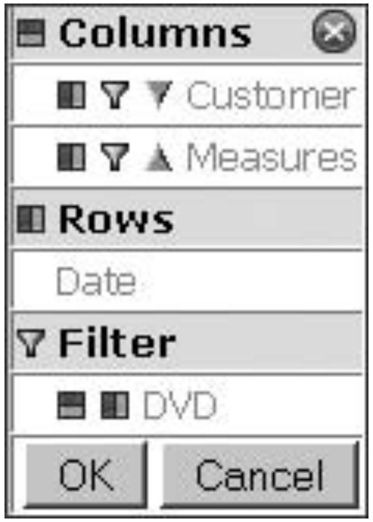
			Measures
Date	DVD	Customer	● Revenue
⊞ All Date.Monthss	⊞ All DVDs	⊞ All Customers	¤10,921,326.03

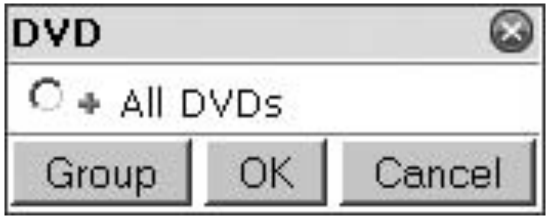
Slicer:

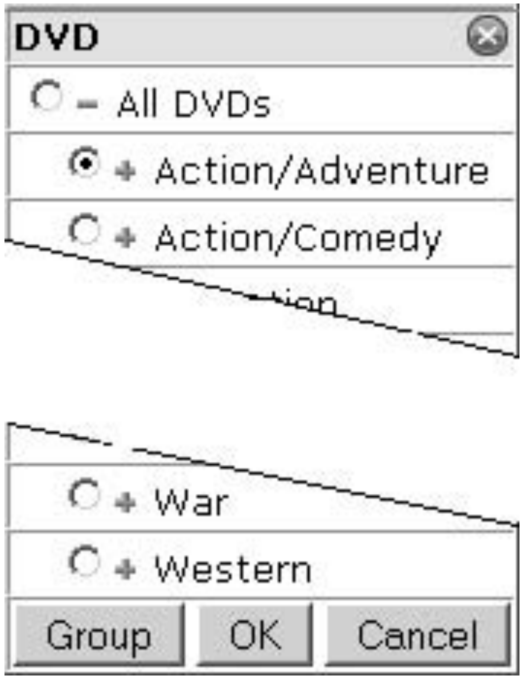
The OLAP navigator

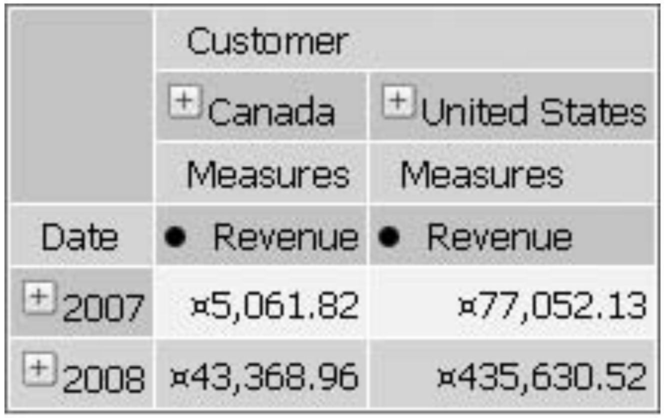


Slicing with the OLAP Navigator





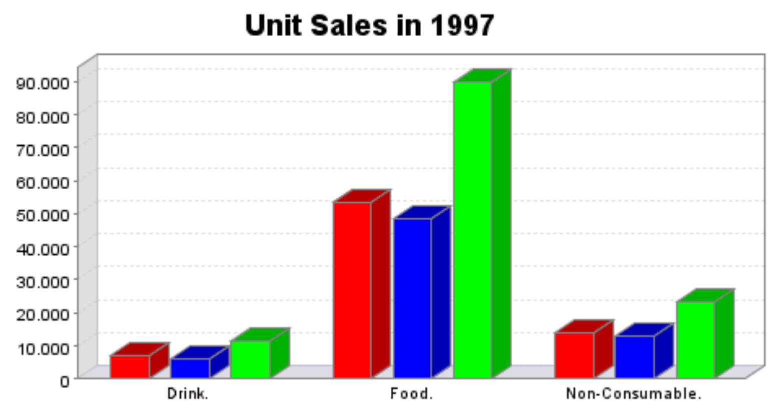




	Customer	
	Canada	United States
	Measures	Measures
Date	● Revenue	● Revenue
2007	¤5,061.82	¤77,052.13
2008	¤43,368.96	¤435,630.52

Slicer: [Genre=Action/Adventure]

Chart



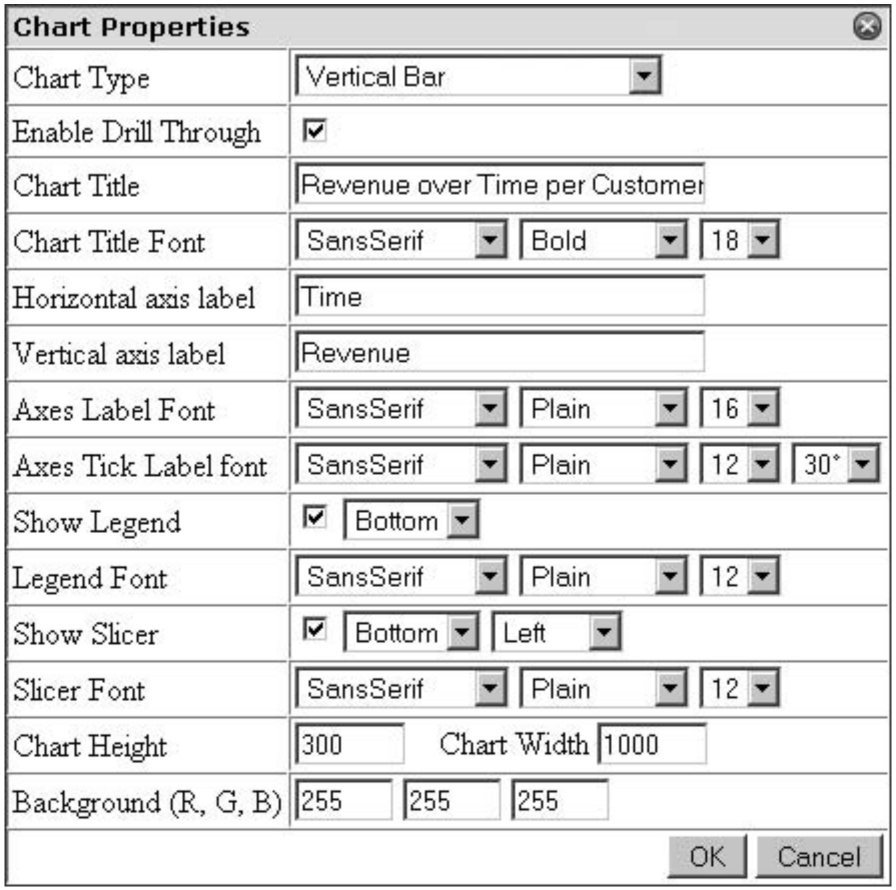


Chart Properties

Chart Type	Vertical Bar
Enable Drill Through	✓
Chart Title	Revenue over Time per Customer
Chart Title Font	SansSerif, Bold, 18
Horizontal axis label	Time
Vertical axis label	Revenue
Axes Label Font	SansSerif, Plain, 16
Axes Tick Label font	SansSerif, Plain, 12, 30°
Show Legend	✓ Bottom
Legend Font	SansSerif, Plain, 12
Show Slicer	✓ Bottom, Left
Slicer Font	SansSerif, Plain, 12
Chart Height	300 Chart Width 1000
Background (R, G, B)	255 255 255