

## ETL design training session – basic

This document serves as a training session for using Pentaho Data Integration (PDI) tool (a.k.a. Kettle) for designing an ETL process that answer user analytical needs (expressed in a form of textual requirements).

### Content:

- a. In the first part of this training, we will briefly explain the difference between data flow (i.e., transformation) and control flow (i.e., job) in Pentaho Data Integration and how to create them.
- b. During the second part, we will use an example textual requirement (requirement 1) to walk you through the process of creating an ETL data flow design using the Pentaho Data Integration tool.
- c. Lastly, in the third part, we will create a control flow for managing the execution of our ETL process.

**Data sources used in the training:** All examples are created over the Learn-SQL system. For better understanding of internals of the system under the study, the schematic/diagrammatic representations of the domain ontology and the available data sources are available (see Sources/Explanations.pdf):

- 1) Diagrammatic/schematic representation of the Learn-SQL system
- 2) Moodle DB schema (IE notation), which captures the Learn-SQL database schema
  - a. Follow the instructions in 'Sources/DB Moodle/ DBConnectionSetup.pdf' to set up the database connection and deploy the LearnSQL database.
- 3) Candidate's information in XML format, which is provided by Moodle
- 4) Evaluation results (candidates with final marks) in excel (dni, mark), which is the Excel sent to FIB at the end of the course

# ETL design: training session - basic

## Part A: Data vs. Control Flow

In ETL design, we primarily differentiate between data and control flow.

- **Data flow** is in charge of performing various operations over data themselves in order to prepare them for loading into a DW (or directly for exploiting them by end users). That is, data **extraction** (reading from the data sources), various data **transformation** tasks (data cleaning, integration, format conversions, etc.) and finally **loading** of the data to previously created target data stores of the DW. Data flows are typically executed as a **pipeline** of operations (rather than a set of strictly sequential steps), meaning that they all start executing at the same time but keep idle until they receive data at the input. In PDI, data flows are called **transformations**, they have extension .ktr, and they are created as follows: **File -> New -> Transformation**
- **Control flow**, on the other side is responsible of orchestrating the execution of one or more data flows. It does not work directly with data, but rather on managing the execution of data processing (i.e., scheduling, starting, checking for possible errors occurred during the execution, etc.). Unlike data flow, in control flow the order of execution is strictly defined by the sequence of activities, meaning that one activity does not start its execution until all its input activities have finished. This is especially important in the case of dependent data processing, where the results of one data flow are needed before starting the execution of another. In PDI, control flows are called **jobs**, they have extension .kjb, and are created as follows: **File -> New -> Job**

## Part B: Create an ETL data flow (transformation) that answer the textual requirement

In this exercise, we will simulate the creation of an example ETL process with a single user requirement. Notice that while ETLs typically require several complex data flows (managing the loading of both factual and dimensional tables), here we practice on a single data flow that prepare data to answer a single user requirement (i.e., to create a multidimensional cube that can answer such requirement).

### Requirement 1:

*“Analyze the average number of colisions (**response\_processing\_colisionsATRIBUT**) from the point of view of the candidate’s department (**candidate\_departmentATRIBUT**) and year (**year\_month\_yearATRIBUT**) for the assessment items with difficulty level (**assessment\_item\_difficultyATRIBUT**) greater than 7. “*

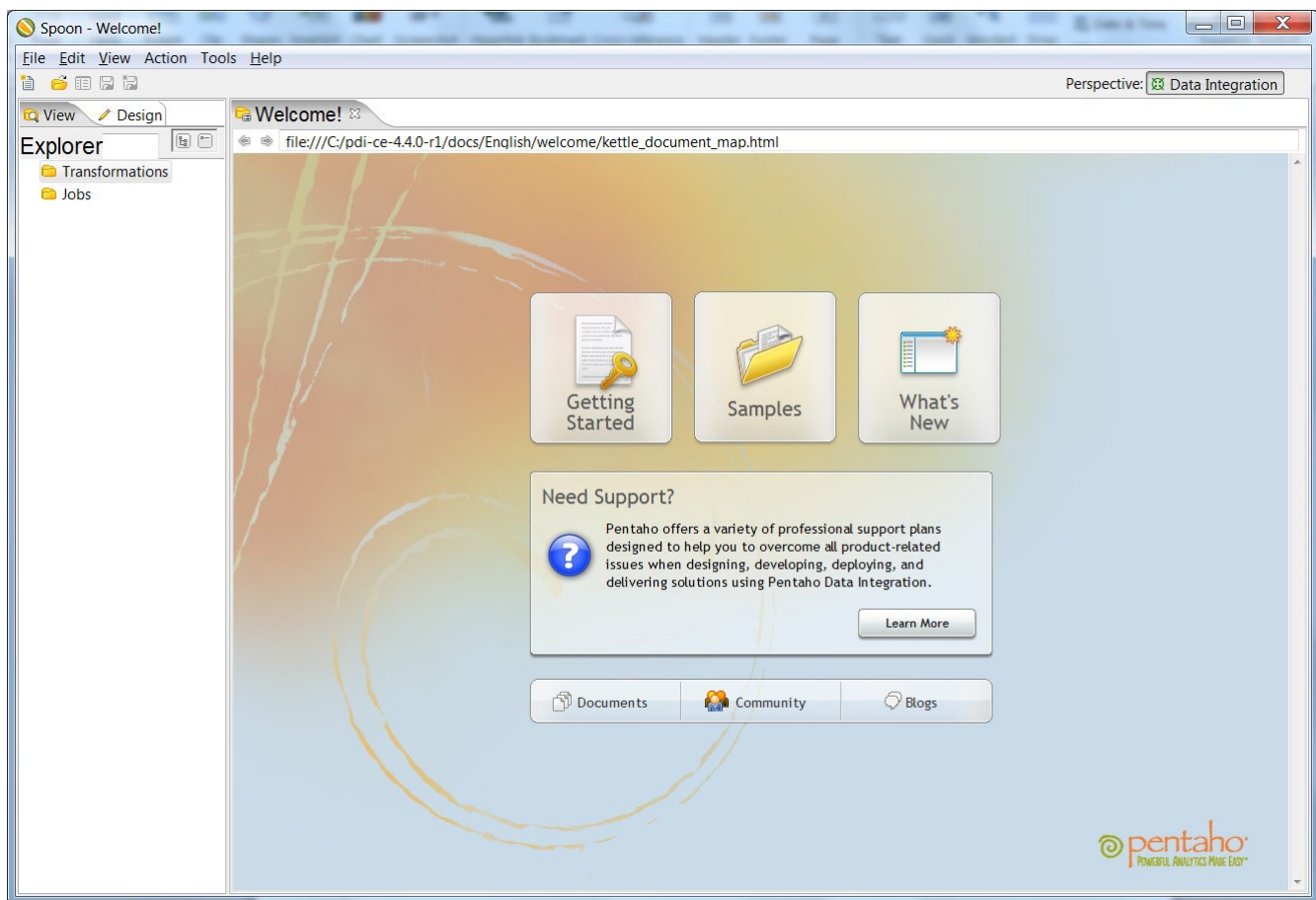
### Design process:

#### 1. Opening Pentaho’s graphical ETL editor (Spoon):

The ETL design in Pentaho Data Integration tool should be provided using their graphical editor (Spoon).

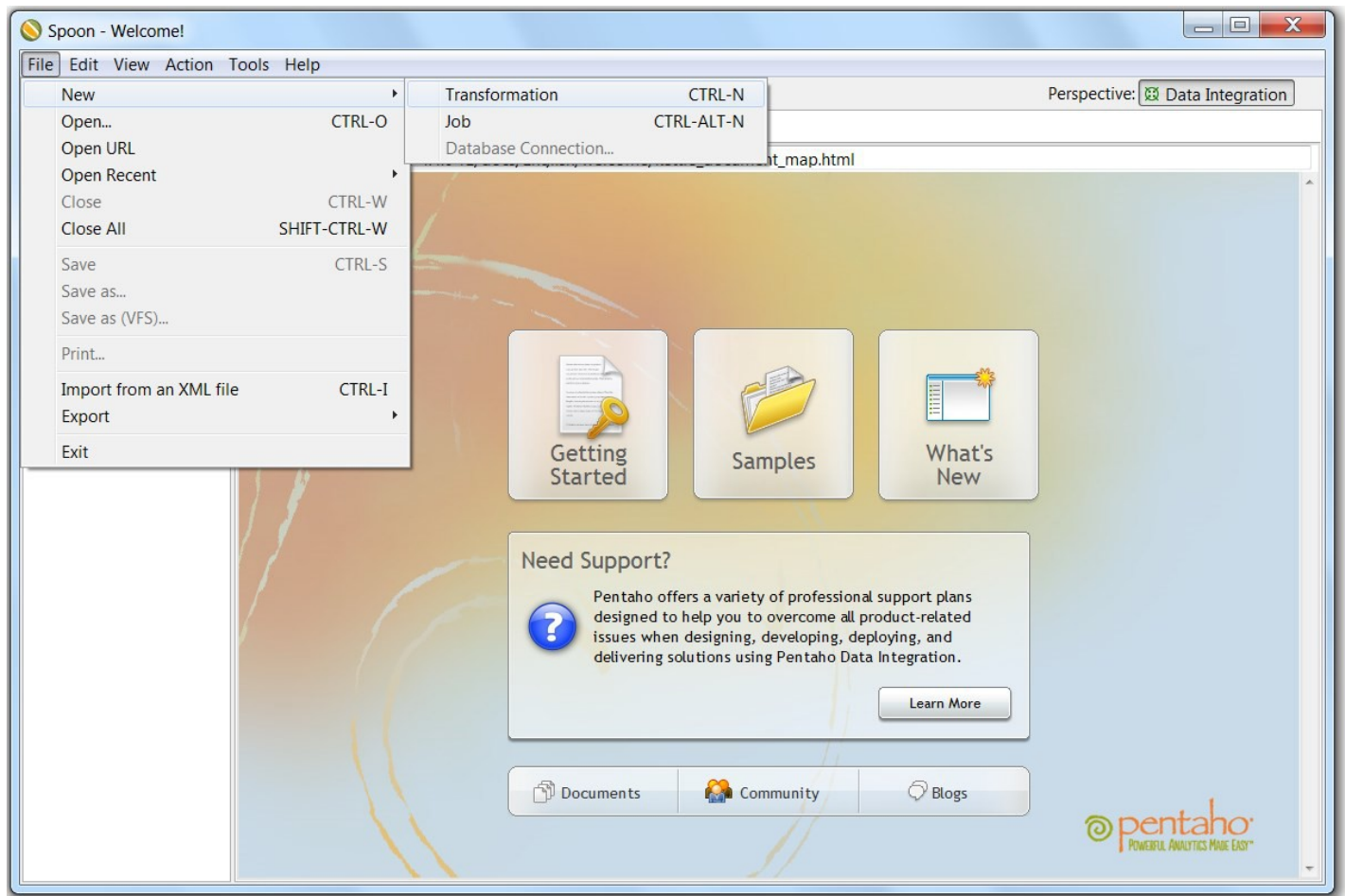
Inside the Pentaho Data Integration distribution package (PDI) under the data-integration folder we start PDI graphical editor by opening **Spoon.bat** (for Windows users) or **Spoon.sh** (for UNIX users).

After we open Spoon tool, we are supposed to see the following starting screen (please notice that depending on the version you are running, the welcome screen can differ):

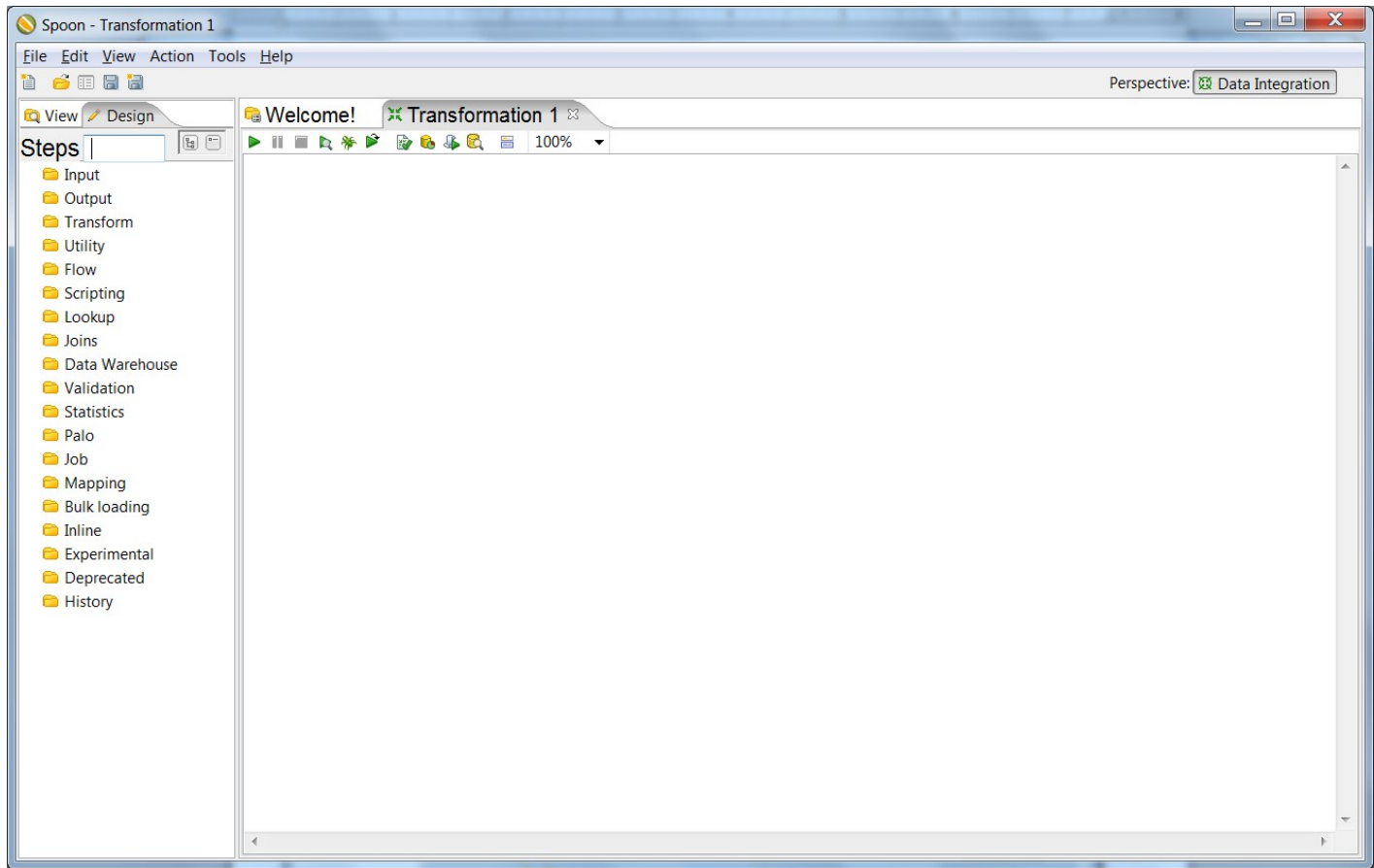


## 2. Creating empty transformation:

To create new transformation (for designing an ETL flow) we choose **File->New->Transformation** as shown in the figure below.



After we created a new transformation the empty design canvas should be visible in the screen as shown in the figure below.



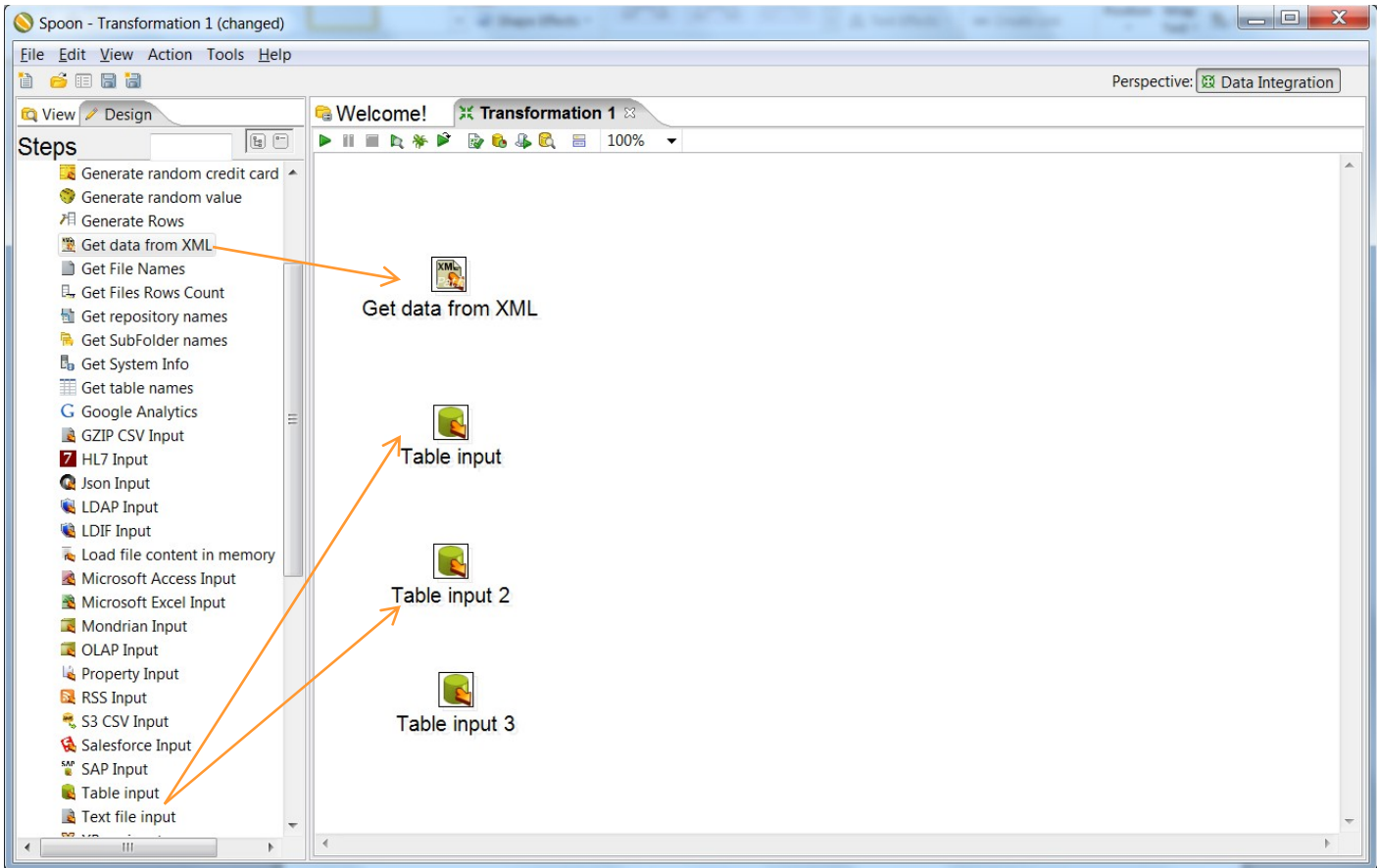
### 3. Defining data source inputs of the ETL process:

First, we need to identify the sources from where the ETL process is supposed to read the data. Considering the learnSQL system (detailed in Sources/Explanations.pdf) three different sources are available **1)** LearnSQL DB. **2)** Candidate's information in XML format. **3)** Evaluation results (candidates with final marks) in excel (dni, mark).

Considering our example we identify the following mappings:

- ***response\_processing\_colisionsATRIBUT** maps to the **staging\_area\_resultats\_sw** table of LearnSQL DB and its column **collision**.*
- ***candidate\_departmentATRIBUT** maps to the **mdl\_user.xml** file and the entries located at XPath **/mdl\_user/row/department***
- ***year\_month\_yearATRIBUT** maps to the **staging\_area\_resultats\_sw** table of LearnSQL DB and its column **momentinsert** with the sql function applied to extract year form the timestamp: **extract (year from momentinsert) as year***
- ***assessment\_item\_difficultyATRIBUT** maps to the **t\_questions** table of LearnSQL DB and its column **difficultat**.*

For each of these mappings we need to define the data input step. Input steps templates are provided in the design template located at the left side of the Spoon window in the category "Input". Drag and drop the templates need in the Spoon canvas as shown in the figure below.



**a) Parameterize table input step:**

After we import the table input step template to the canvas can parameterize it by double clicking the step icon in the canvas. The following form will open:

(1) First, we may define the name of the step by entering desired name in the Step name text box.

The screenshot shows the 'Table input' configuration dialog box. The 'Step name' field is highlighted with a red box and labeled '(1)'. The 'Connection' field is empty. The 'SQL' field contains a template query: `SELECT <values> FROM <table name> WHERE <conditions>`. Below the SQL field, there are checkboxes for 'Enable lazy conversion' and 'Replace variables in script?'. The 'Insert data from step' field is empty. The 'Execute for each row?' checkbox is unchecked. The 'Limit size' field is set to 0. The dialog box has 'OK', 'Preview', and 'Cancel' buttons at the bottom.

- (2) Then we need to define the connection to the DB from where the data should be read by choosing the option “New”. Set the connection parameters as in the figure below. (*Enter your username and password instead*).

Table input

Step name: response\_processing\_colisionsATRIBUT

Connection: [Dropdown] [Edit...] [New...]

SQL: `SELECT <values> FROM <table name> WHERE <conditions>`

Line 1 Column 52

Enable lazy conversion ☐

Replace variables in script? ☐

Insert data from step [Dropdown]

Execute for each row? ☐

Limit size: 0

[OK] [Preview] [Cancel]

- (3) To test if we properly configured the database connection click “Test” and if everything is alright we should be able to see the following message.

Database Connection

General

Connection Name: oracleDB

Connection Type: Oracle

Access: Native (JDBC)

Settings:

Host Name: oraclefib.fib.upc.edu

Database Name: orabd

Tablespace for Data

Tablespace for Indices

Port Number: 1521

User Name: username

Password: \*\*\*\*\*

[Test] [Feature List] [Explore]

(3)

Database Connection Test

Connection to database [oracleDB] is OK.

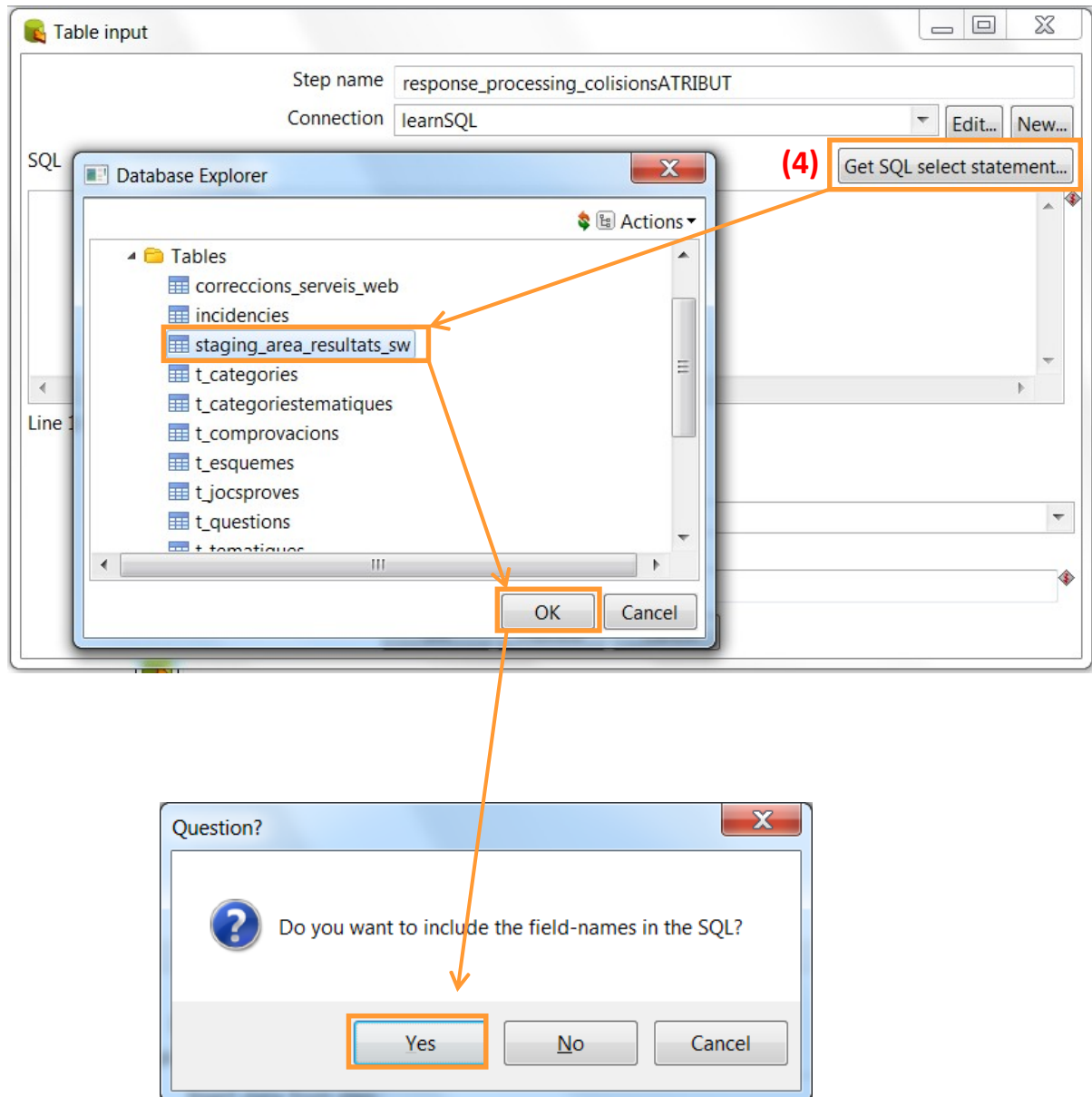
Hostname: oraclefib.fib.upc.edu

Port: 1521

Database name: orabd

[OK]

- (4) After we configured the database connection we can either manually define the sql statement needed to extract the data or automatically obtain it by choosing “Get SQL select statement”. The example for extracting data from **staging\_area\_resultats\_sw** table is shown in the figure below.





- (5) The SQL statement is automatically created and it includes all the columns found in the *staging\_area\_resultats\_sw* table.

The screenshot shows the 'Table input' dialog box. The 'Step name' is 'response\_processing\_colisionsATRIBUT' and the 'Connection' is 'learnSQL'. The 'SQL' text area contains the following statement:   
`SELECT  
 moodle  
 idusuari  
 idquestioremota  
 nomjp  
 momentinsert  
 resultatcodi  
 duradacorreccio  
 duradaobtencioresultat  
 nomcompr  
 collisions  
 closed_session  
 semester  
FROM staging_area_resultats_sw`  
A red '(5)' is placed next to the first column 'moodle'. To the right of the SQL text area is a button labeled 'Get SQL select statement...'. Below the SQL text area, the 'Line 1 Column 0' is indicated. There are several checkboxes: 'Enable lazy conversion' (unchecked), 'Replace variables in script?' (unchecked), 'Insert data from step' (dropdown menu), and 'Execute for each row?' (unchecked). A 'Limit size' field is set to '0'. At the bottom are 'OK', 'Preview', and 'Cancel' buttons.

- (6) As we do not need the data of all the columns to be read we can remove ones not needed and/or apply operations to extract parts of other attributes (e.g., `extract (YEAR from momentinsert)`). Finally we get the following SQL statement:

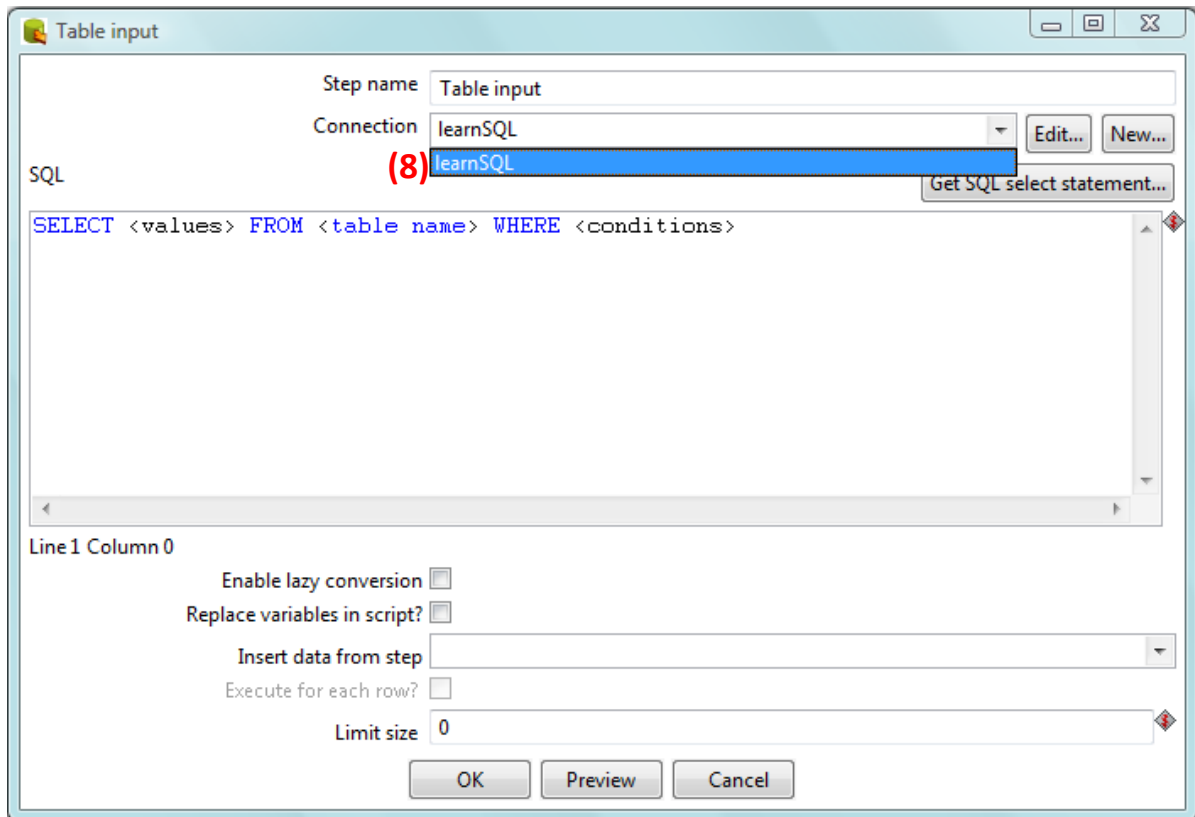
The screenshot shows the 'Table input' dialog box with the same 'Step name' and 'Connection'. The 'SQL' text area now contains:   
`SELECT  
 collisions, idusuari, idquestioremota  
 , extract (YEAR from momentinsert) as year_month_year  
FROM staging_area_resultats_sw`  
A red '(6)' is placed next to the new column. Below the SQL text area, the 'Line 2 Column 37' is indicated. The checkboxes and 'Limit size' field are the same as in the previous screenshot. At the bottom, a red '(7)' is placed next to the 'OK' button.

Note that we included additional columns as they are needed to map the relations of the response\_processing to the candidate (idusuari), assessment\_item (idquestioremot).

- (7) Finally we confirm the definition of the table input step for response\_processing by choosing "OK".

Similarly, we add other table input steps into the canvas and parameterize them.

- (8) For example **assessment\_item\_difficultyATRIBUT** is extracted from the same database. First we choose the database connection we created before as shown in the figure below.



- (9) Next, following the same procedure we create the SQL statement to extract **assessment\_item\_difficulty** *ATTRIBUTE* as shown in the figure below. Note that we also need to extract the id (primary key) of the *t\_questions* table in order to relate it to the response\_processing concept (i.e., the *staging\_area\_resultats\_sw* table).

Table input

Step name:

Connection:

SQL

```
select
  id
, difficultat
from t_questions
```

(9)

Line 1 Column 0

Enable lazy conversion ☐

Replace variables in script? ☐

Insert data from step

Execute for each row? ☐

Limit size:

**Important note:** Notice that various operations that are to be performed inside the ETL flow can be as well supported by the source engine (in this case Oracle DBMS). For example, filtering, various format conversions, operations over temporal attributes (date/time difference, interval overlapping, etc.). Therefore, for the sake of optimizing data processing, and to avoid loading more data than needed into an ETL engine, we can push those operations to the source engine. In that case, they should be implemented within the SQL script in Table Input step as showed above.

**b) Parameterize Get data from XML step:**

- (1) First, we may define the name of the step by entering desired name in the Step name text box.

Get XML Data

(1) Step name:

File | Content | Fields | Additional output fields

XML source from field

XML source is defined in a field? ☒

XML source is a filename? ☐

Read source as Uri ☐

get XML source from a field

File or directory

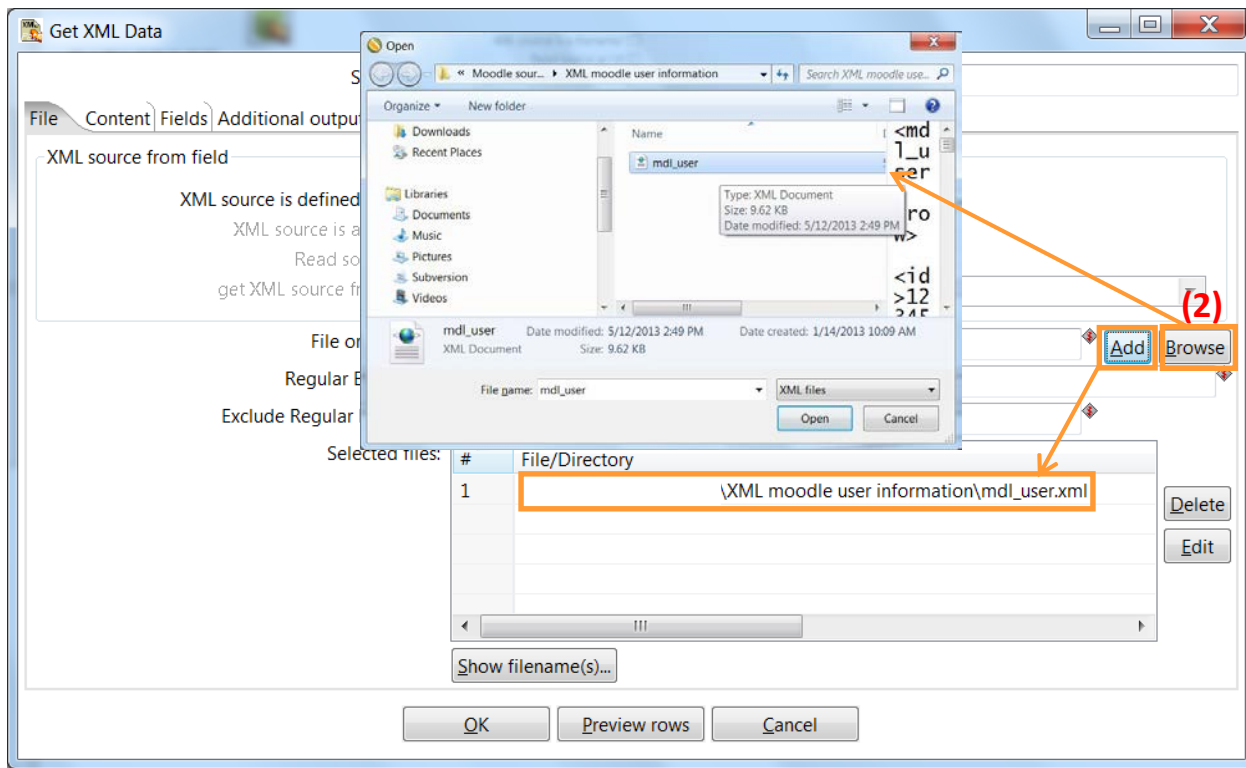
Regular Expression

Exclude Regular Expression

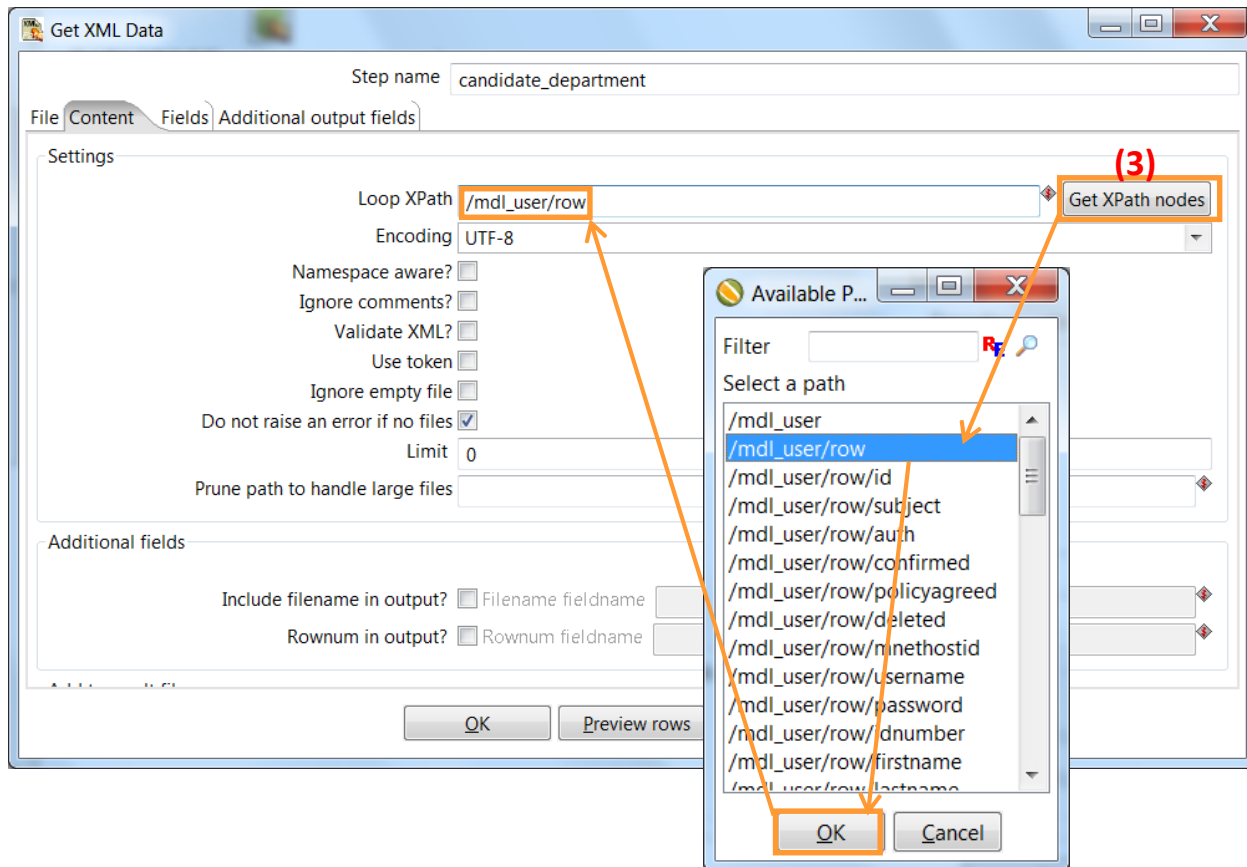
Selected files:

#	File/Directory	Wildcard (RegExp)	Exclude wildcard	Required
1				

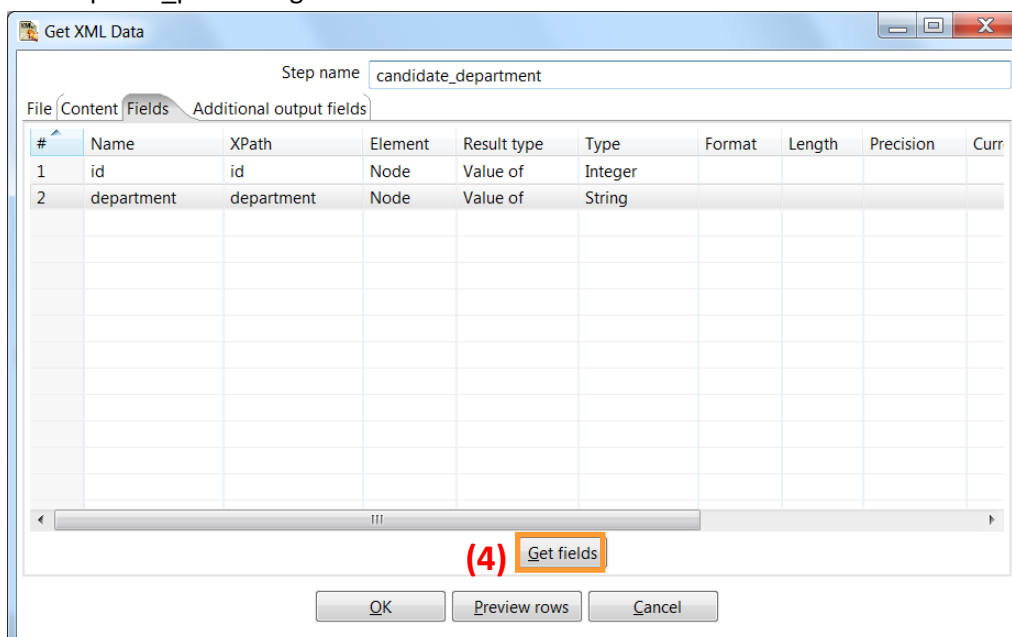
(2) You should then in the “File” tab of the dialogue define the filepath to the xml file as show in in the figure below.



- (3) We should then in the “Content” tab of the dialogue define the XPath from where inside the mdl\_user.xml file the data should be read. We can choose it from automatically obtained XPath nodes – “Get XPath nodes”, as shown in the figure below.



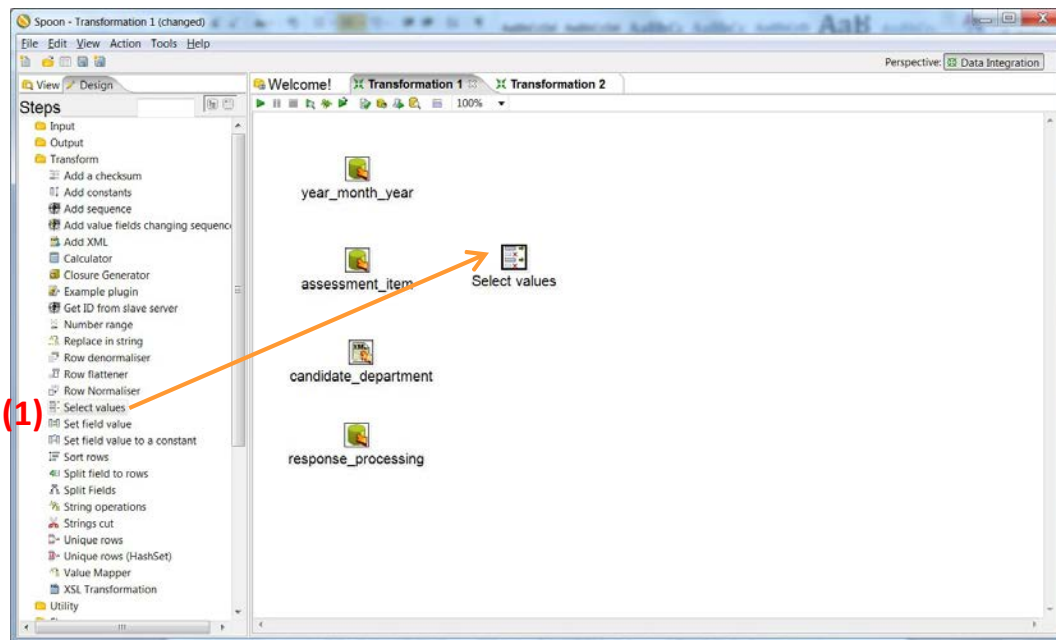
- (4) We should then in the “Fields” tab of the dialogue define the fields to be read from the input xml file. We can first automatically obtain all the fields from the defined XPath (/mdl\_user/row) by choosing the option “Get fields” and then remove unnecessary ones. Finally the following fields are needed to be extracted. See figure below. Notice that we left the identifier of a candidate entry in order to relate the candidates to the response\_processing.







#### 4. Adding the processing steps of the ETL process:


After we defined the input data source steps to extract data we need to define the processing steps of the ETL process.

- (1) First, for each input data store step we need to add a rename step that will assure that all the field names in the process are unique. This step is important as various independent sources may have the same names for the fields with different semantics. From the “Transform” category of the design palette we choose “Select Values” step and drag it to the Spoon canvas. See figure below.



- (2) To insert the hop from the `assessment_item` data store to the select values step we first hover the mouse over the table input step (`assessment_item`) until the step editing options appear as in the figure below. Four different options are available:

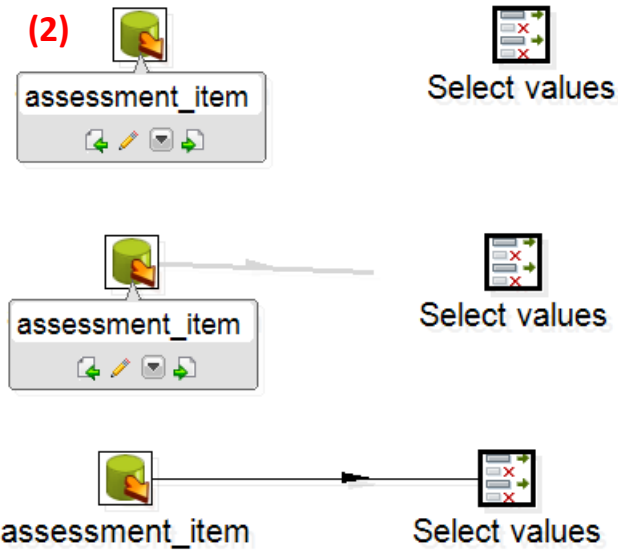
-  - input step connector: Connects this step to its producer step.
-  - edit the properties of the step: Opens the edit step dialogue for this step. (Alternative to double click on the step icon.)
-  - Opens the option list of this step. (Alternative to the right-click on the step icon)
-  - output step connector: Connects this step to its consumer step.

We then press and hold the last option () and drag the hop to the select values step as shown in the figure below. Most of the hops are added in the same manner.

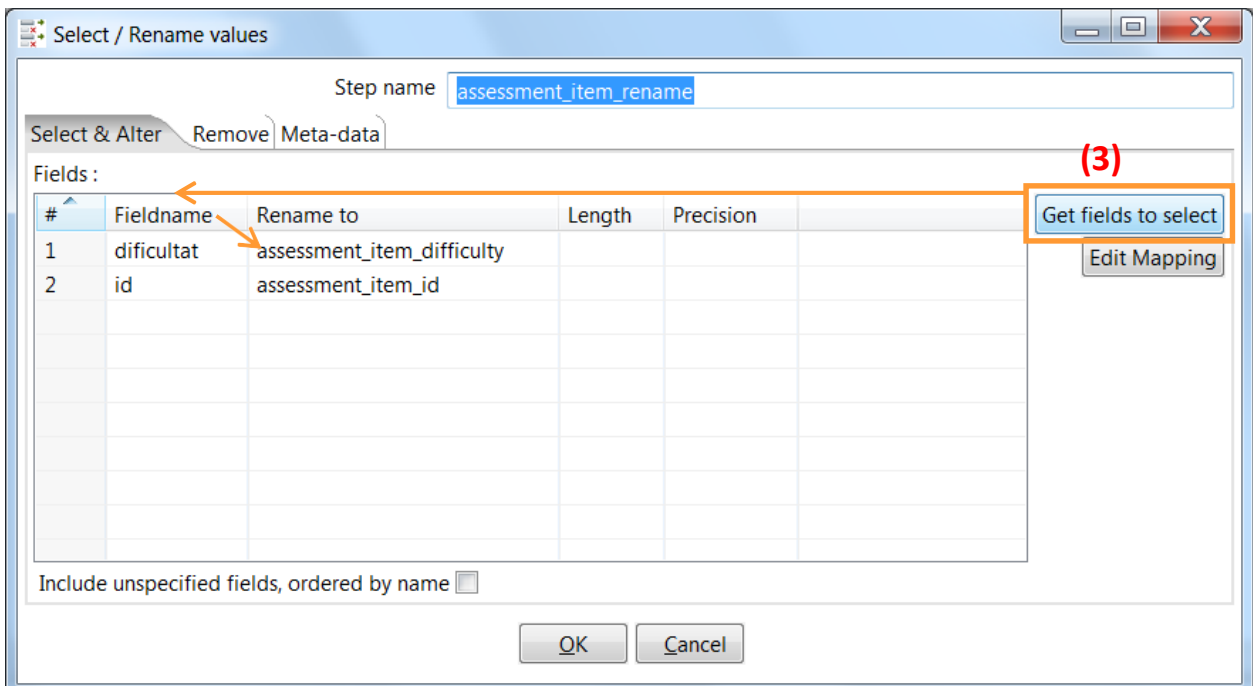
The exceptions are:

- i) Hops from “Select Values” where we can specify if the hop is the main output of the current step or the hop to error handling step. In this exercise always specify the hop as the “Main output of the step”.
- ii) Hops from “Filter Rows” where we can specify if the hop passes the data that fulfills the filter condition “Result is TRUE”, does not fulfills the filter condition “Result is FALSE”, or the main output of the step. In the first two cases both true and false hops must be specified or the exception will occur during execution. In this exercise always specify the hop as the “Main output of the step”.

Remember to first add the hop from the previous step to the step you are currently inserting so you can get the information (available fields) from the previous step.

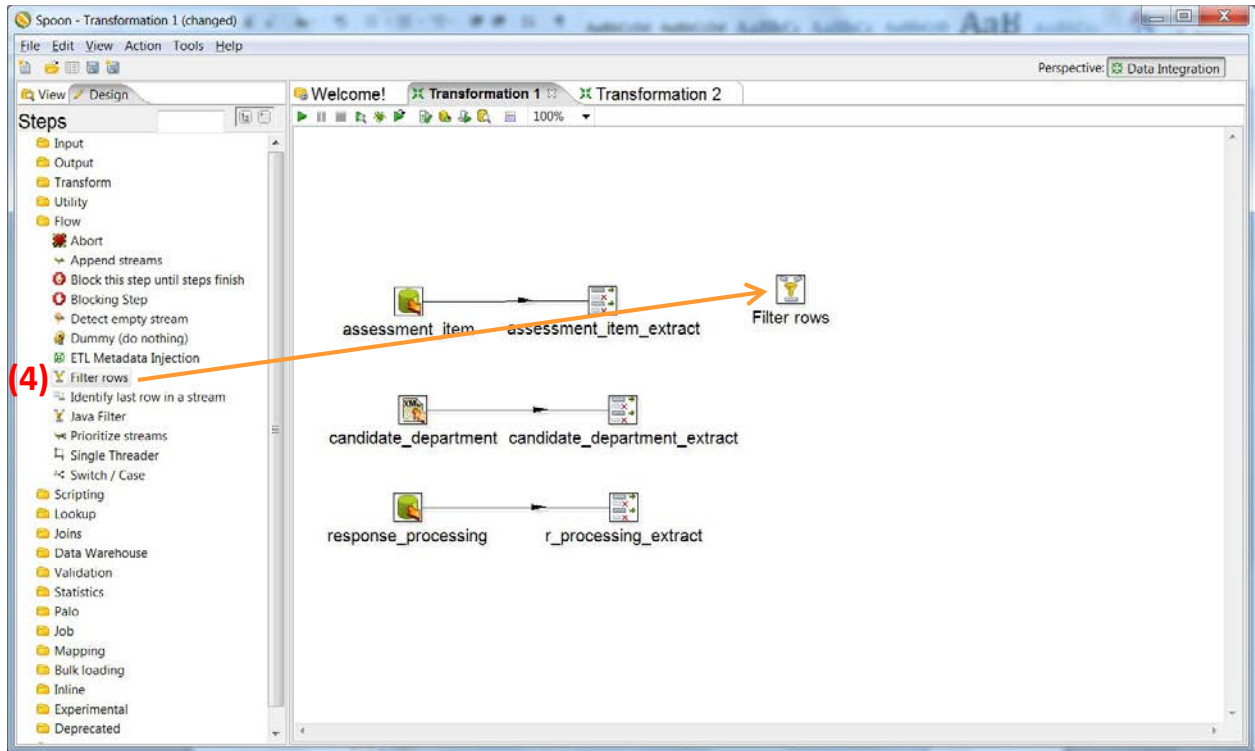


- (3) To parameterize the select values step we can automatically obtain the fields from the input step by choosing the “Get fields to select” option and then in the column “Rename to” we can enter the new name of the given field. Note that in order to propagate the field to the next step we need to list it in the “Select&Alter” tab of the dialogue below, even if we do not rename it (i.e., “Rename to” value is optional). See figure below. Rename steps for all other inputs are defined in the same manner.

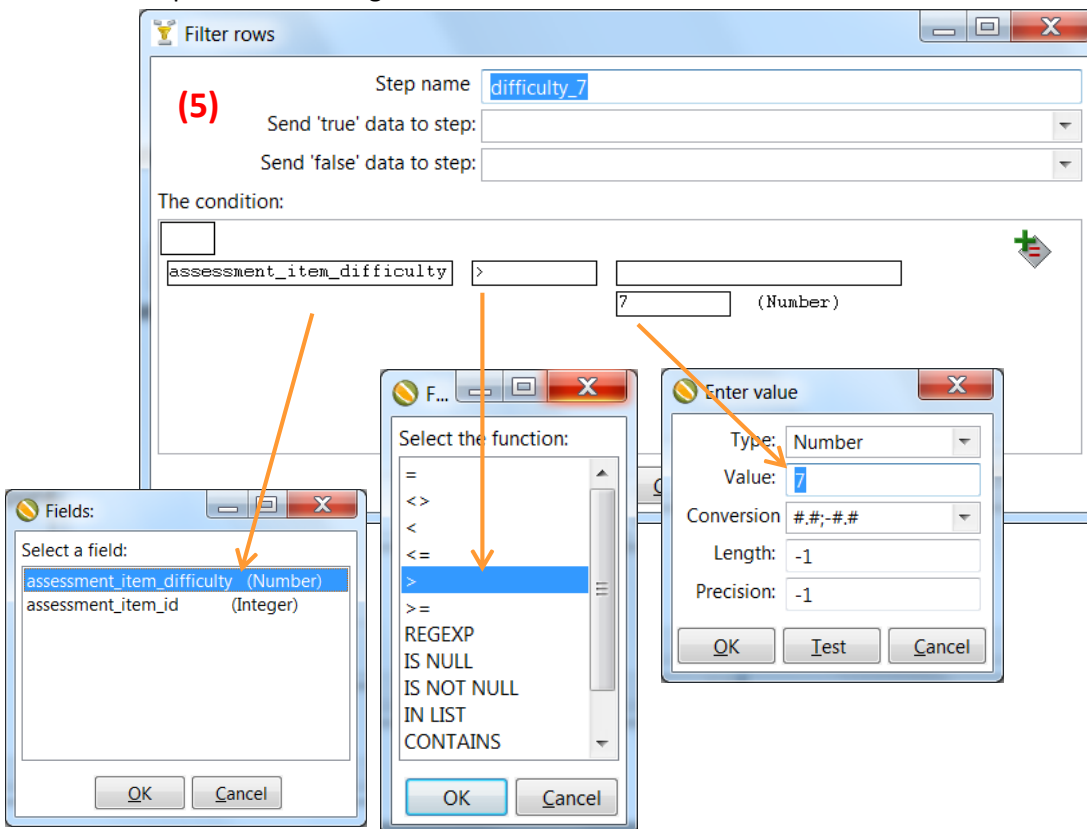


**Important note:** the Select values step, besides renaming can also be used to alter the metadata of input fields (i.e., data type, length, precision, or format of the data). To do this, you should open the tab “Meta-data”.

- (4) Then, we need to add filtering steps for the descriptor defined in the requirement. From the “Flow” category of the design palette choose “Filter rows” template and drag and drop it to the Spoon canvas as shown in the figure.



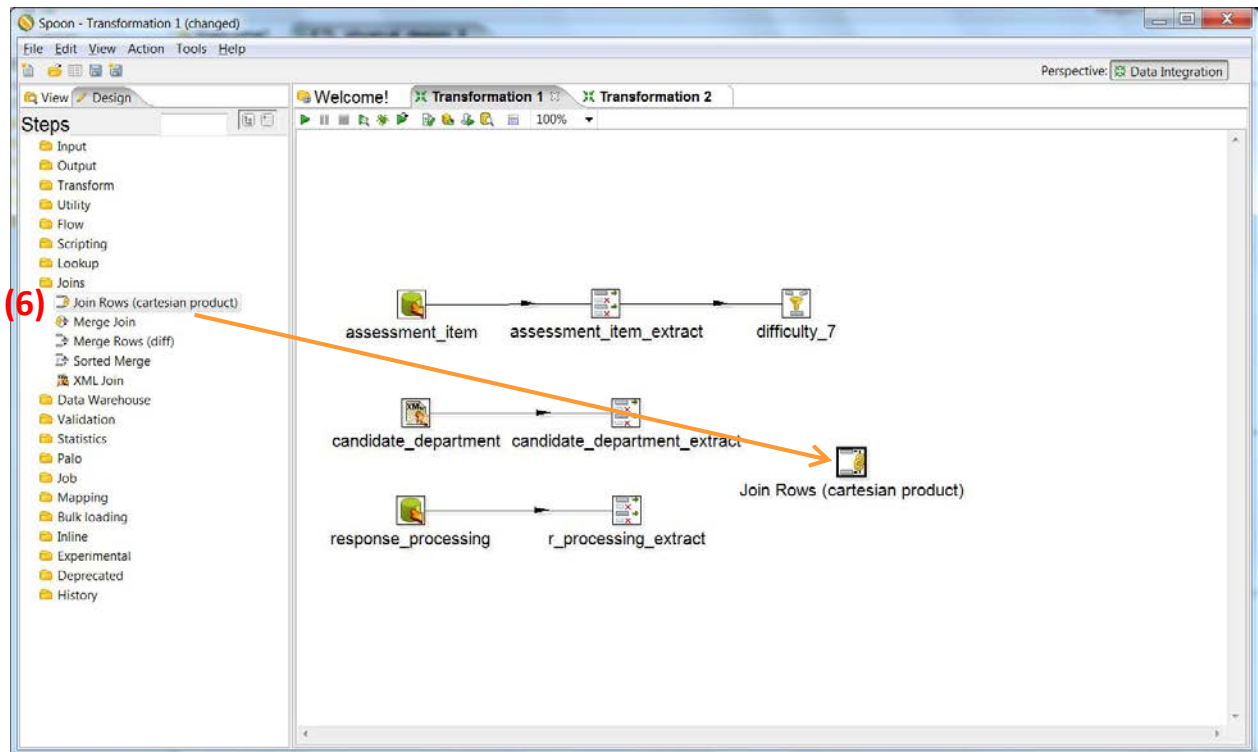
- (5) We need to parameterize the filter rows step considering the descriptor provided in the input requirement. See figure below.



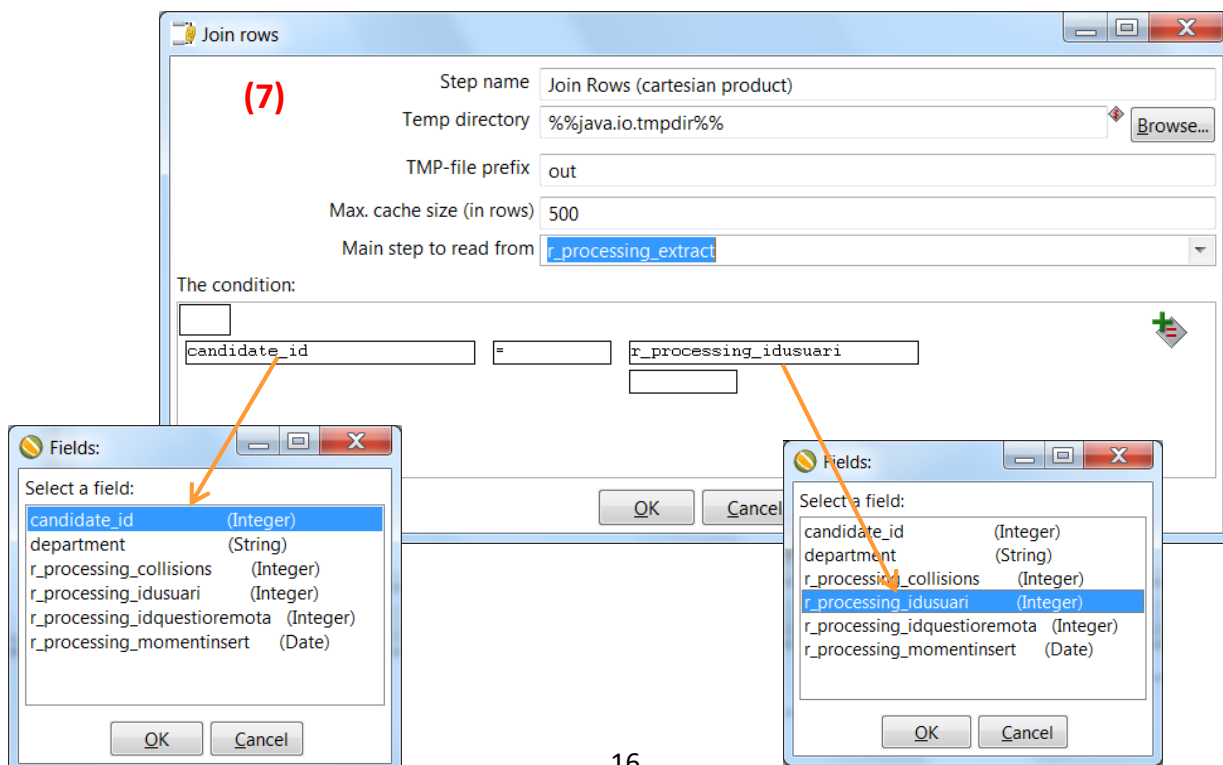
**Important note:** while Filter rows provides you with typical predefined predicates to filter on the values of input fields, to implement more complex Filtering operations, you can use Java Filter (from the same Flow palette of steps – see Figure above).



- (6) Then we add the Join operations to relate data coming from the defined input sources. From the “Joins” category of the design palette we choose “Join Rows” step template and drag it to the Spoon canvas as shown in the figure below.

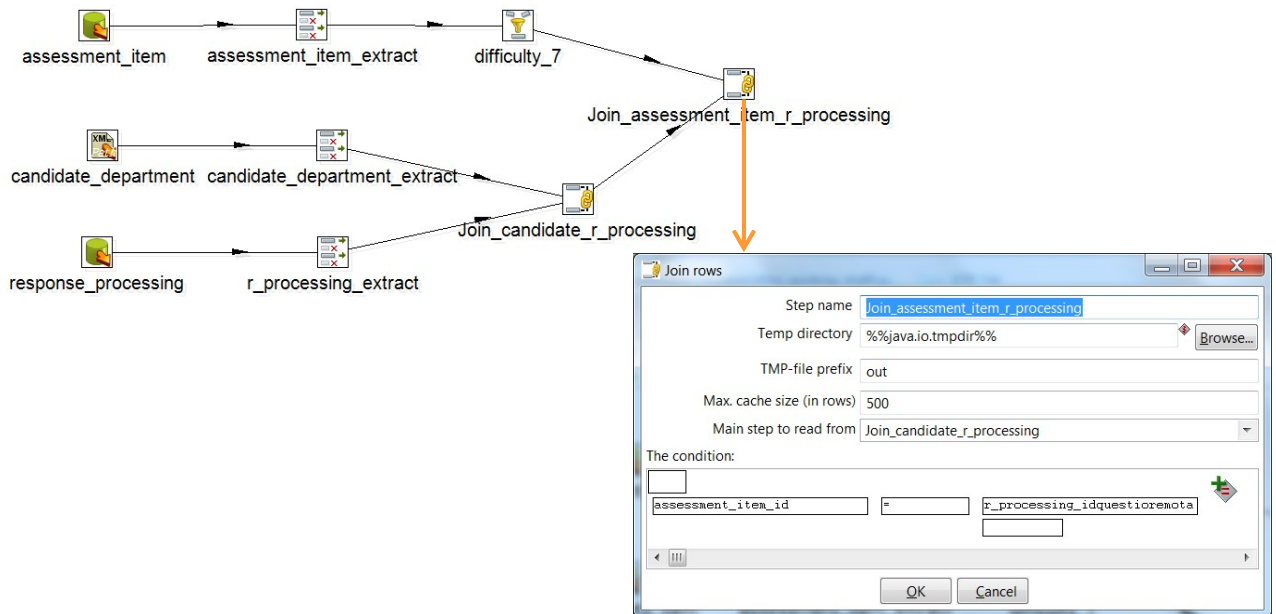


- (7) Then after connecting data source steps to the added join step we should parameterize the Join Rows step to properly relate in this case candidate\_department\_extract and response\_processing\_extract steps. We define the “Main step to read from”, which defines the previous step from which we read most of the data, while the data from the other step can be cached to the disk. Also, we need to define the join attributes. In the case of this join these are “candidate\_id” from the candidate\_department input and “r\_processing\_idusuari” from the response\_processing input. See figure below.

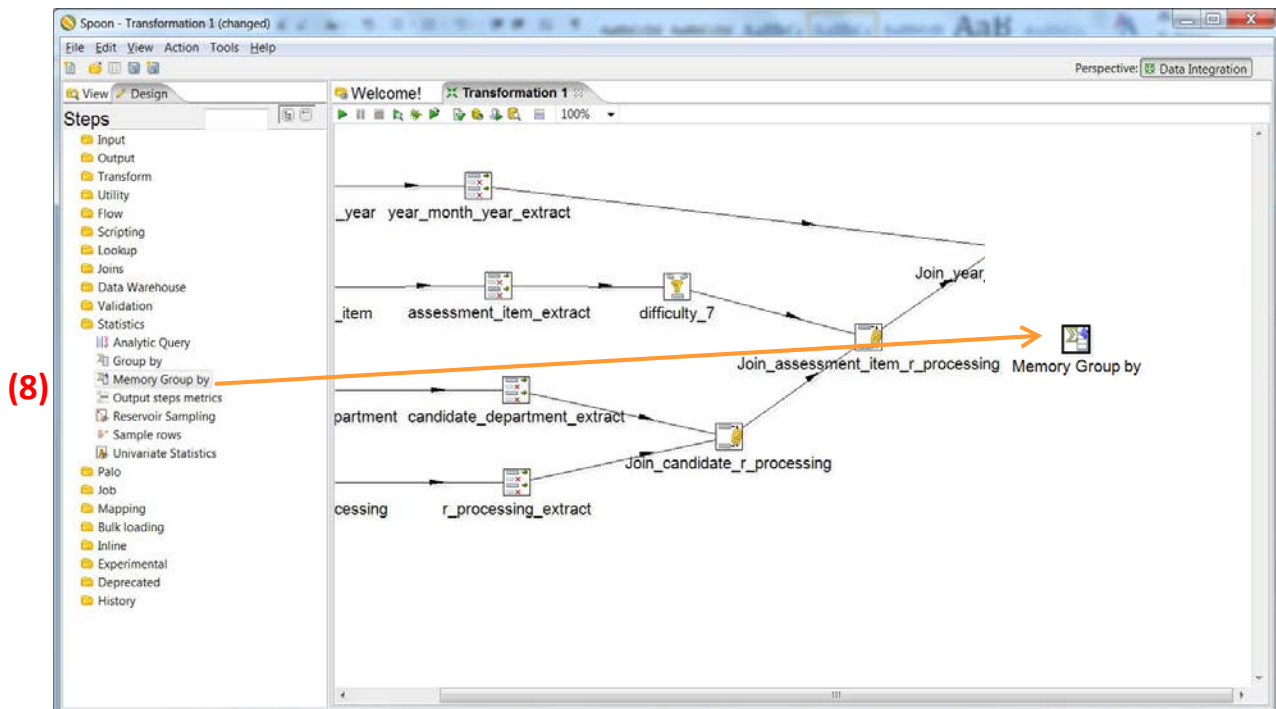


Iteratively, in the similar manner we define other joins among the rest of the inputs. We connect the steps and set the join attributes of the other joins as shown in the figures below:

- **assessment\_item** and **response\_processing**



- (8) After we related all the inputs we need to aggregate data considering the given requirement. From the “Statistics” category of the design palette we choose “Memory Group By” step template and drag it to the Spoon canvas as shown in the figure below.



- (9) We then parameterize the memory group by step considering the requirement. First we define grouping attributes. We automatically obtain the fields from the previous step by choosing “Get Fields” option and removing unnecessary fields or simply by choosing the fields from the drop down list (see upper part in the figure below). Following the requirement we choose year\_month\_year and department as grouping attributes. Then we define aggregation function. We first enter the output name of the derived aggregated measure (avg\_collisions) and then from the drop down lists we choose the subject field and the required aggregation function. In our case we need to perform average over the response processing collisions (see the down part of the figure below)

Step name: Memory Group by

Always give back a result row: ☐

The fields that make up the group:

#	Group field
1	year_month_year
2	department

Aggregates :

#	Name	Subject	Type
1	avg_collisions	r_processing_collisions	Average (Mean)

OK Cancel

(10) After we create the ETL process that answers the requirement we can run and test the flow by choosing the run icon (green play button) and launching the flow locally with the default parameters as shown in the figure below.

**Execute a transformation**

Local, remote or clustered execution

☒ Local execution

☐ Execute remotely

☐ Execute clustered

Remote host:

☐ Pass export to remote server

☒ Post transformation

☒ Prepare execution

☒ Start execution

☐ Show transformations

Details

☐ Enable safe mode

☒ Clear the log before execution

Log level: Basic logging

Replay date (yyyy/MM/dd HH:mm:ss):

Parameters

#	Parameter	Value	Default value
1			

Variables

#	Variable	Value
1	Internal.Job.Filename.Directory	Parent Job File Direct
2	Internal.Job.Filename.Name	Parent Job Filename
3	Internal.Job.Name	Parent Job Name
4	Internal.Job.Repository.Directory	Parent Job Repository

Arguments

#	Argument	Value
1		

**Launch** **Cancel**

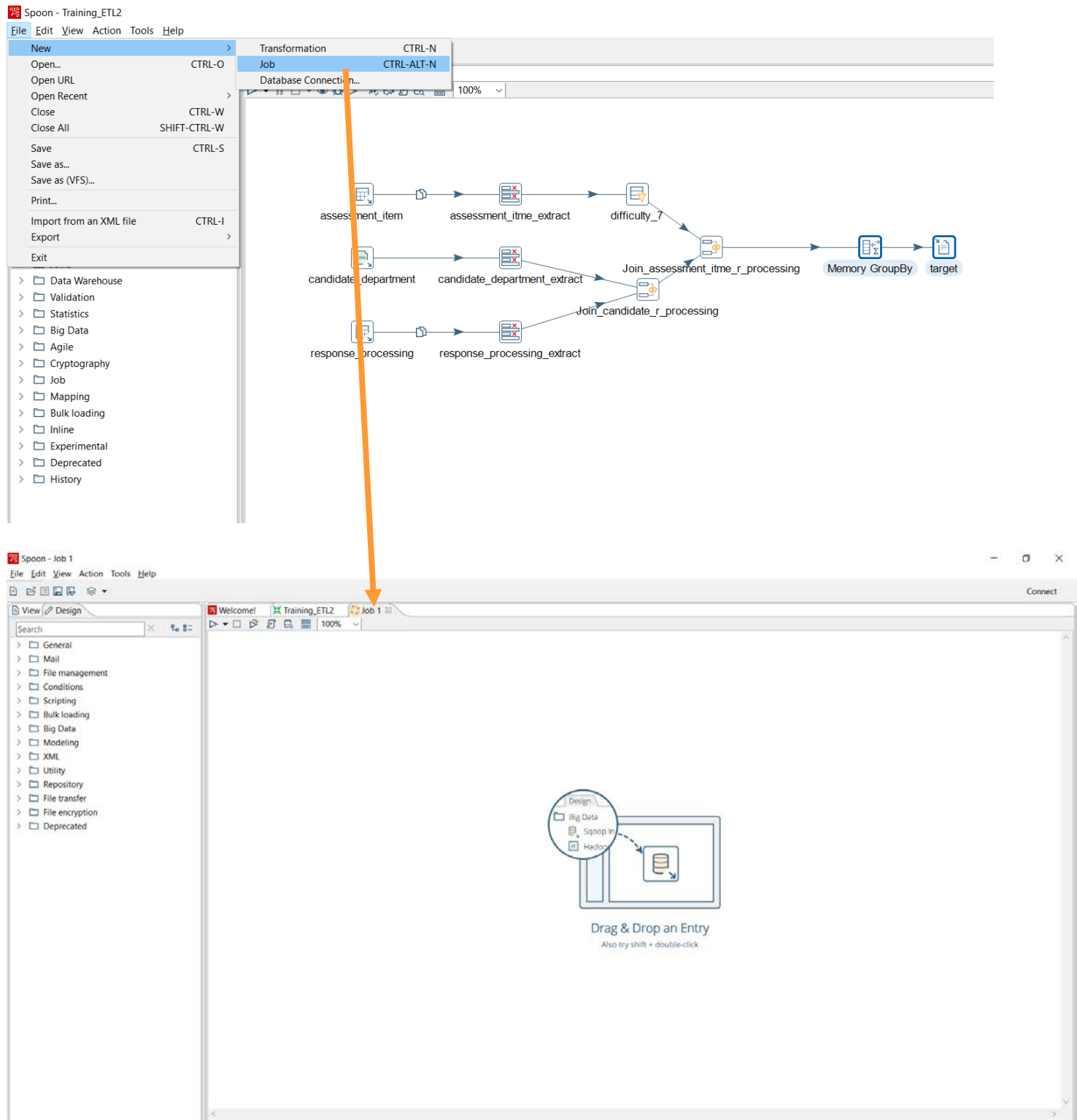
**Execution Results**

Execution History | Logging | Step Metrics | Performance Graph | Metrics | Preview data

#	Stepname	Copynr	Read	Written	Input	Output	Updated	Rejected	Errors	Active	Time
1	response_processing	0	0	4000	4000	0	0	0	0	Finished	12.4s
2	response_processing_extract	0	4000	4000	0	0	0	0	0	Finished	12.3s
3	candidate_department	0	0	400	400	0	0	0	0	Finished	0.3s
4	assessment_item	0	0	300	300	0	0	0	0	Finished	2.4s
5	candidate_department_extract	0	400	400	0	0	0	0	0	Finished	0.5s
6	assessment_item_extract	0	300	300	0	0	0	0	0	Finished	2.3s
7	difficulty_7	0	300	37	0	0	0	0	0	Finished	2.3s
8	Join_candidate_r_processing	0	4400	3540	0	0	0	0	0	Finished	12.5s
9	Join_assessment_item_r_processing	0	3577	354	0	0	0	0	0	Finished	12.6s
10	Memory GroupBy	0	354	1	0	0	0	0	0	Finished	12.6s
11	target	0	1	1	0	2	0	0	0	Finished	12.6s

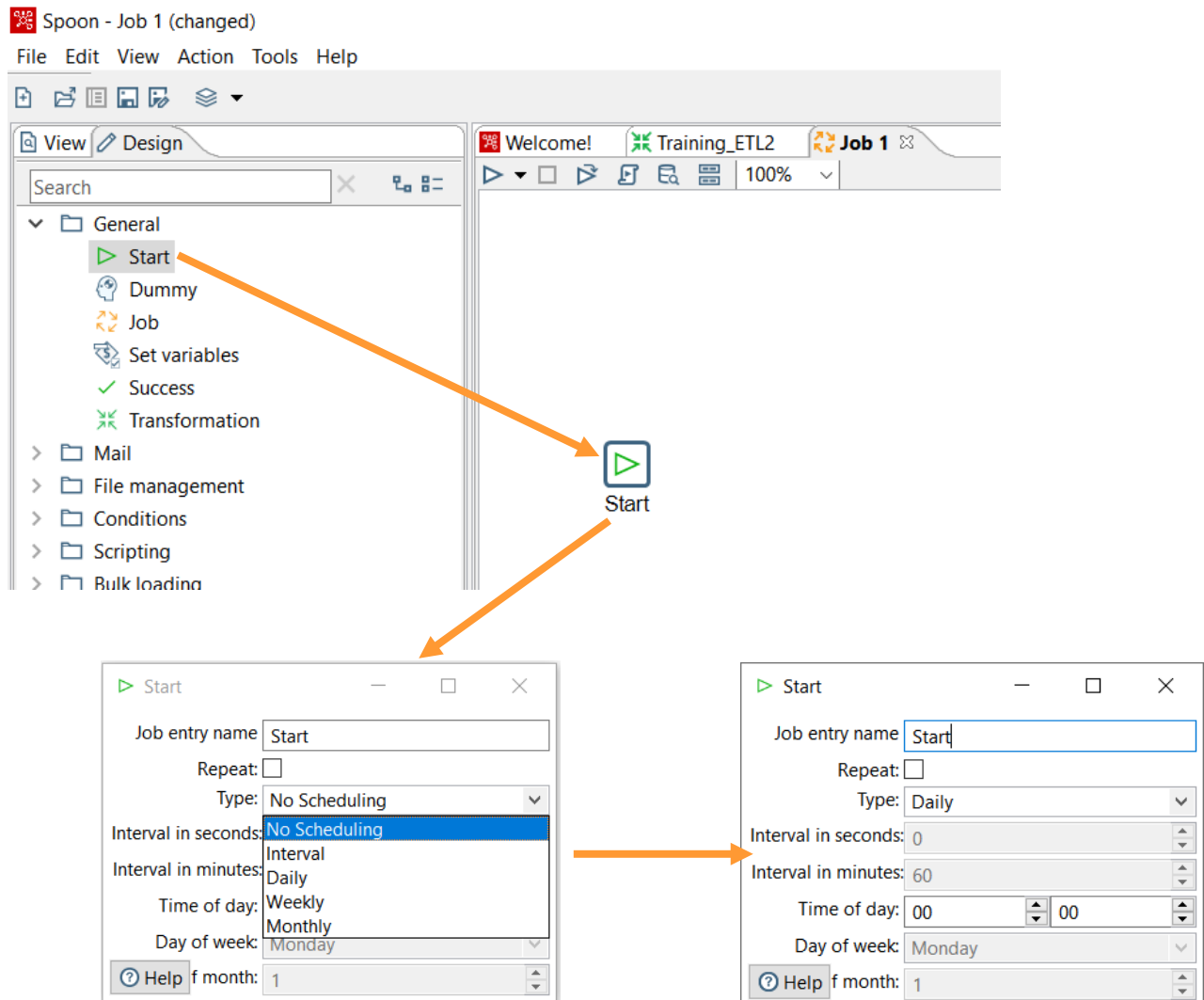
## Part D: Create control flow (job) to execute an ETL process

Once we have our data flow(s) created, we need to manage their execution, i.e., to define the order in which they will be executed, in order to respect dependencies among them (e.g., first loading dimension tables then fact table). To that end, as explained in Part A, we will create a simple PDI job (control flow).



### (1) Start activity

From the job design palette on the left, select General -> Start to introduce a start activity into a control flow.

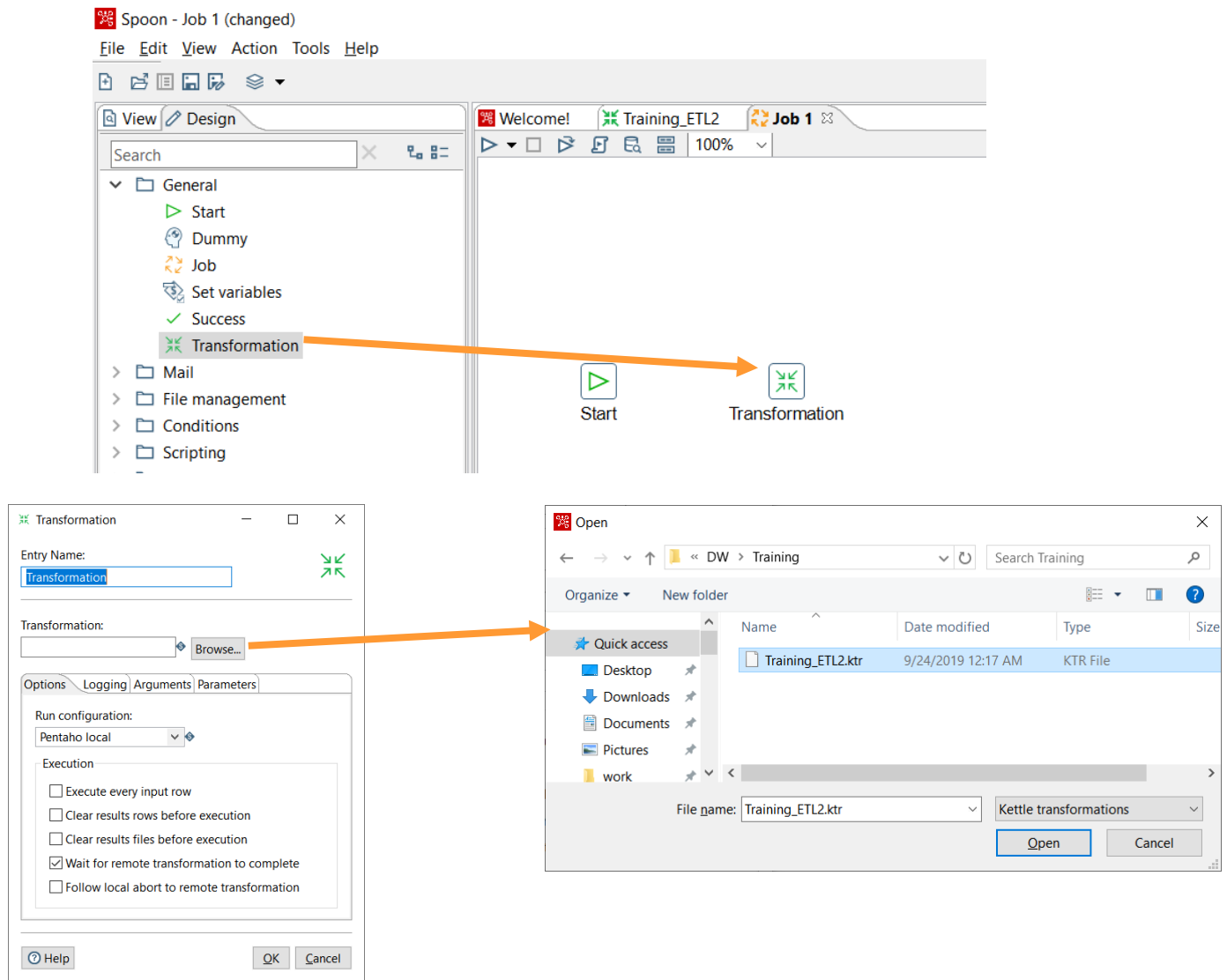


The Start activity allows us to define the possible scheduling for starting the execution of an ETL flow (i.e., within a certain defined time interval, daily, weekly, or monthly). In addition, we can also set the repetition of the execution (e.g., daily at exactly a certain time of day, like midnight).

### (2) Transformation activity

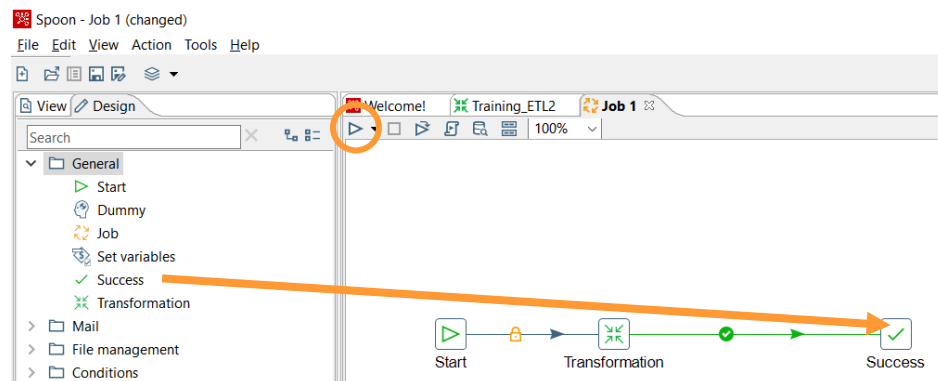
Next, one of the most used activities is the Transformation activity, that triggers the execution of the previously created data flow, i.e., transformation.

The activity is again selected from the job design palette on the left (category: General). Next, we need to browse and select the previously created transformation (.ktr) that will be executed within this job activity (see Figure below). Besides that, we can configure the activity with different options for execution, logging file where the execution logs will be stored, possible arguments that are required for the execution (e.g., internal variables), and also obtaining the parameters from the previous activity and passing it to the next transformation activity in the job.



### (3) End activity – success

Finally, we add an activity that ends the execution of the job having that it finished successfully (Success). Notice that while this activity does not have any additional configuration, it is just used to control the successful end of the ETL execution.



To run and test an ETL control flow locally press play button in the palette above.

We finalize our ETL design training here, but notice that Pentaho Data Integration supports various complex job activities (e.g., running externally Hadoop or Spark jobs and reusing the resulting data within the same ETL flow). We invite you to play with the PDI tool and define more complex data and control flows for practice.