aSISt

Data mapping module / Snapshot module / Data source

FINGO Team

version: 5.64.0.0   04.2025

Table of Contents

Table of Figures

1 Introduction

aSISt facilitates creation of various financial reports and makes it possible to fit in into various data schemes. For this purpose it delivers advanced mechanisms allowing source data download from different bank systems (so-called back office) and data processing into a form required by the supervising entity.

Depending on internal solutions adopted by the bank, source data can be derived from one or multiple independent systems (data warehouse, transactional systems, sale systems, etc.). For this reason, in order to prepare a final financial report, it is necessary to e.g. combine data prior to processing or perform adequate transformations.

The actions executed, from the moment data are downloaded from the source system until the final report preparation, are called a reporting process. This process, as defined above, is presented on Figure 1.

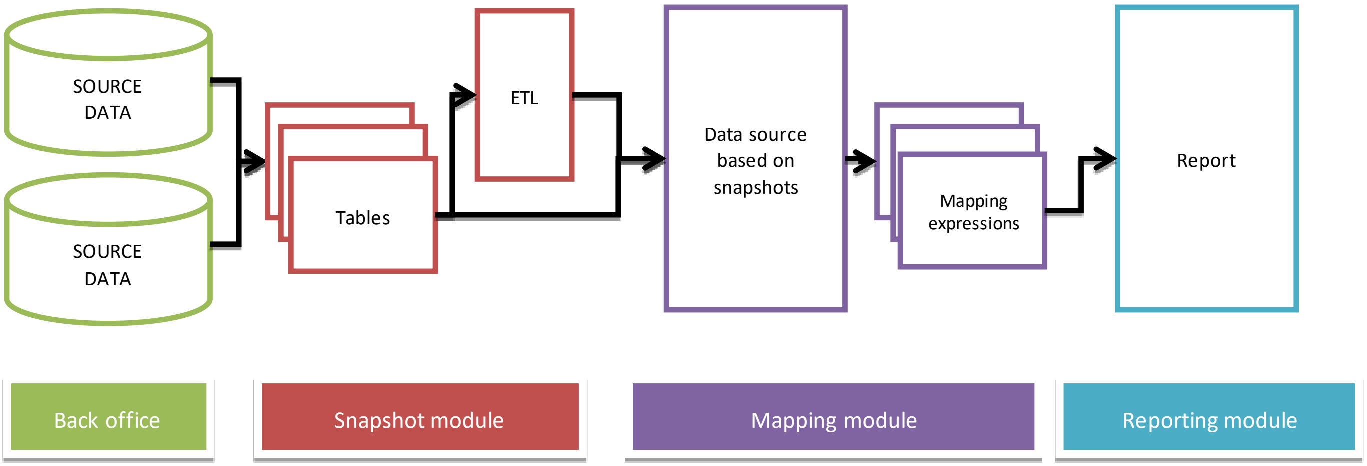


Figure 1. Reporting process layout

In the first stage of the reporting process data are in the bank's back office systems.

Second stage: with the use of "Snapshot module", the appropriate source data are transferred to the embedded into aSISt "Source data repository".

When this stage is being prepared it may turn out that data will require transformations, resulting from the necessity to filtrate, aggregate or select data appropriately. aSISt has an embedded ETL model with the use of which the user can perform all of these operations.

EFL processing is defined at the "Snapshot module" stage.

The third stage consists of defining data sources, mapping sets and mapping expressions with the use of "Mapping module" and then fetching data into the prepared reports.

The last stage of the reporting process is performed in the "Reporting module" and is based on reconciling reported data, their validation in accordance with the binding control rules, and generation of the result file.

"Data mapping" and "Snapshot warehouse" modules are additional **aSISt** functionalities which facilitate fetching data from many various data source to the prepared reports.

They allow for:

- creation of snapshot definitions based on tabular data and with the use of the embedded into aSISt ETL-type tool (ETL – Extract, Transform and Load) which transforms data from source systems fetching data to the reports,
- creation of mapping expressions for any reporting fields with which aSISt, with the use of "Fetching by mappings" function, can be filled in with data. Only single CSV, Excel files or advanced results of steps derived from ETL transformations can act as data sources in mapping calculations;
- export of defined mapping expressions and their import to a new set,
- creation of any number of sets for data mapping,
- management of sets for mapping by their deactivation and removal,
- defining various fetching sources and their management.

Defined here:

- **data sources:**
  - are independent from the binding taxonomy,
  - are global for the entire application,
  - can be shared by mapping expressions which use them or by defined snapshots,
  - are a view of single tabular data sources (e.g. CSV file, MS Excel file, SQL query result executed on source data or at subsequent ETL processing stages, etc.).
- **mapping expressions:**
  - are closely linked to "mapping expressions sets",
  - apply to all periods linked with a defined "mapping expression set", referring at the same time to the defined earlier data sources;
  - are used at the last stage of the fetching process,

  - define values of given cells from the prepared report,
- **snapshots:**
  - refer to tabular source data, used in the mapping module as snapshot data sources;
  - can consist of multiple different, independent tables the source of which can be e.g. a CSV file;
  - are not linked with any reporting period,
  - are used in grouping to one tabular data set entity, the data of which can be derived from different systems; such set can be later used for defining a data source.

2 "Data mapping" module

To launch the **"Data mapping"** module select:

⚙ ® Data mapping module

or + – located on the module bookmarks bar and from the highlighted list (see Figure 2) select: Data mapping ...

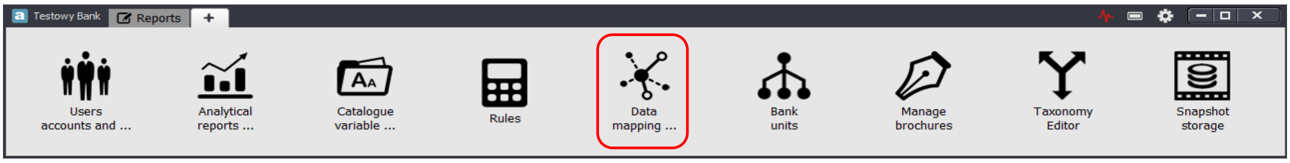


Figure 2. aSISt module bookmarks bar – with the highlighted "Data mapping" module

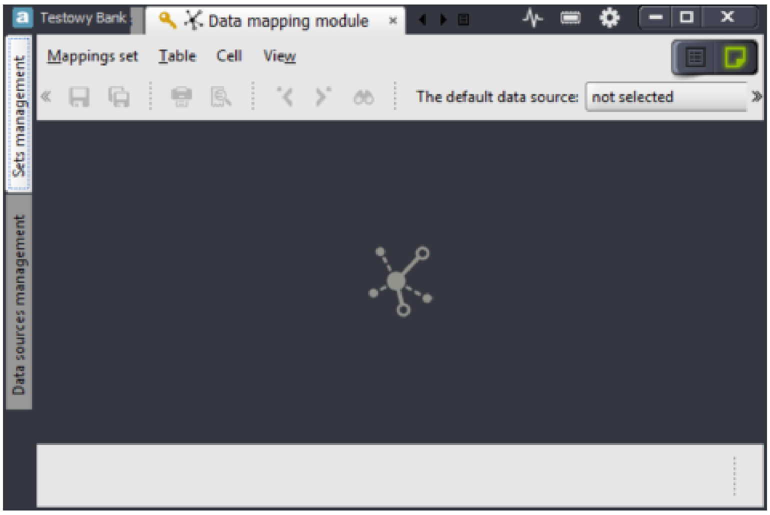
Then a new bookmark Data mapping module × will be created on the module bookmarks bar (see Figure 3), this bookmark groups functions allowing for:

- creation of any number of mapping expressions sets and their management,
- defining mapping expressions and data sources,
- managing data sources for mappings,
- export of defined mapping expressions and their import to another mapping expressions set.

"Data mapping" module:

- is available only for banks with an extended license (including this module),
- in multiple-user version can be simultaneously used only on one workstation.

After "Data mapping module" function is selected, the following (as on Figure 3) appears on screen:



The default data source: not selected

Figure 3. Data mapping module – general view, without an open mapping set

All functions possible to execute are grouped here in two side bookmarks:

- set management (see chapter 2.1 Set management),
- data source management (see chapter 2.2 Data source management)

and described in further chapters.

2.1 Set management

Activation of the **"Data mapping"** module and selection of the **"Set management"** side bookmark (see Figure 3) allows for:

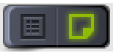
- **creation of new and management of existing mapping expressions sets**, by:
  - opening,
  - deactivation,
  - activation of the previously deactivated ones,
  - verification of setting and details,
  - revision of change history,
- **defining of required mapping expressions**, for the previously created mapping sets, by:
  - input of new and removal of existing expressions,
  - export and import of mapping expressions.

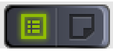
These functions, properly grouped, are available in two modes:

- "Documents" mode – applying to open mapping sets
- "List" mode – applying to previously created mapping sets.

Some functions are available in two modes and others in only one of them.

Depending on the selected work mode, "Document view switch", located in the upper right corner of the screen, can have the following form:

 – when the "Documents" mode is selected

 – when the "List" mode is selected

Switching between those modes is performed automatically by:

- clicking on the greyed-out function button part or
- executing some functions.

2.1.1 Documents mode

"Documents" mode for which the document switch view is as follows is a mode in which all function possible to execute (see Figure 3) are grouped in a menu:

- Mapping set
- Table
- Cell
- View

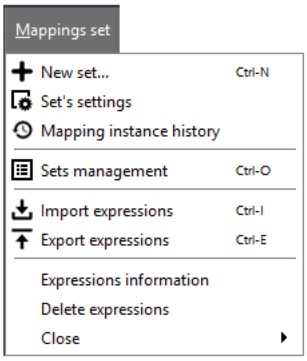
Additionally, the following option was made available:

- Default data source selection

2.1.1.1 Mapping set

The **"Mapping set"** menu (see Figure 4) groups basic **functions connected with mapping sets,** allowing for:

- creation and management of mapping expressions sets,
- introduction of mapping expressions to any cells (including creating mapping expressions for typed dimensions),
- import and export of mapping expressions,
- removal of inactive mapping expressions,
- history review of actions executed in mapping sets.



Mappings set

- New set... Ctrl-N
- Set's settings
- Mapping instance history
- Sets management Ctrl-O
- Import expressions Ctrl-I
- Export expressions Ctrl-E
- Expressions information
- Delete expressions
- Close

Figure 4. "Mapping set" menu in "Documents" mode

2.1.1.1.1 New mapping expressions set

To create **"New mapping expressions set"**, select:

- **"Mapping set" menu** + New set...

or

- icon: +

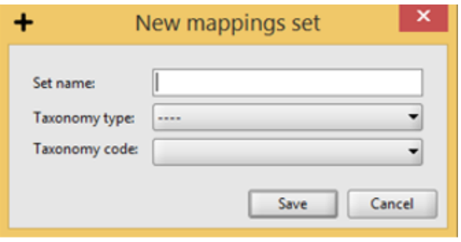
or

- keyboard shortcut: **Ctrl-N**.

This function:

- is available in both work modes, i.e.:
  - in "Documents" mode -
  - in "List" mode –
- allows for creation of new data mapping sets in which mapping expressions will be defined,
- is required in creating separate mapping sets for different taxonomy types,
- considers created mapping expressions sets as mutually independent sets, used subsequently by the "Fetching by mappings" function ***for fetching external data to the reports, according with the defined data source.***

When this function is launched, a window as on Figure 5 is displayed.



New mappings set

Set name:

Taxonomy type: ----

Taxonomy code:

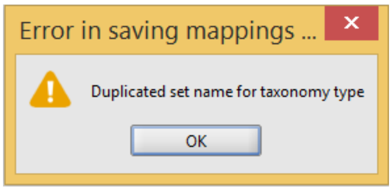
Figure 5. Creating a new mapping expressions set

To create a new mapping expressions set select:

- **set name** – unique name containing a maximum of 20 characters (any permissible),
- **taxonomy type** – select the appropriate report type available from the list displayed, for which mapping expressions will be defined;
- **taxonomy code** – code, existing in the aSISt base, selected from the list displayed.

When entering a set name, the following should be remembered:

- the same name cannot be entered for sets with the same taxonomy type – such sets will not be created (see Figure 6),



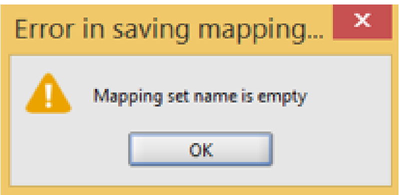
Error in saving mappings ...

Duplicated set name for taxonomy type

Figure 6. Message about a repeated mapping expressions set name for the selected taxonomy type

- the same name can be entered only for different taxonomy types.

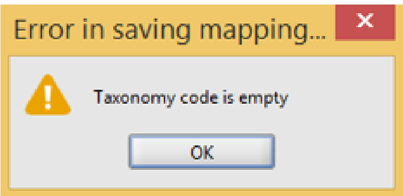
All fields need to be filled in, otherwise a new set will not be created and the user will be notified about this with the appropriate message, as on Figures 7 and 8.



Error in saving mapping...

Mapping set name is empty

Figure 7. Message about an empty name of the mapping expressions set

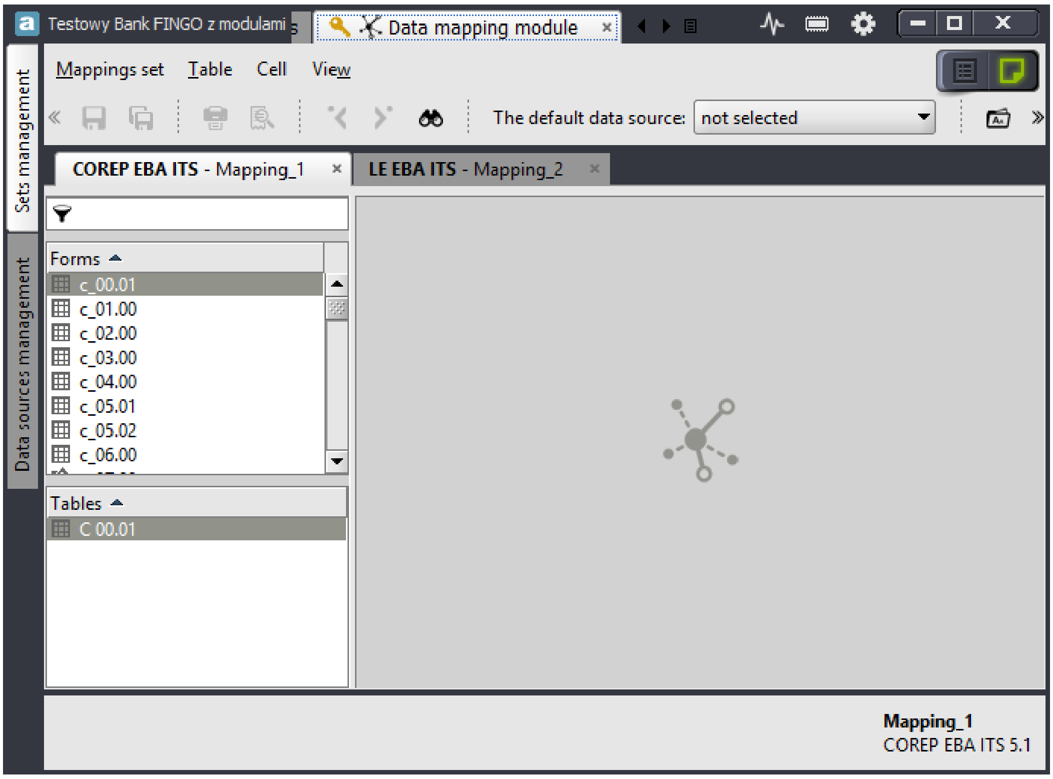


Error in saving mapping...

Taxonomy code is empty

Figure 8. Message about the missing defined taxonomy type for the mapping expressions set

When all required data are entered correctly and confirmed with the button Save, a new required set for mappings will be created and activated and the "Data mapping" module screen will have the form as on Figure 9 (compare with Figure 3).



The default data source: not selected

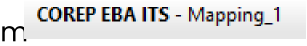
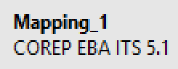
Mapping_1
COREP EBA ITS 5.1

Figure 9. Mapping sets management – documents mode – view with an open mapping expressions set

CREATED MAPPING EXPRESSIONS SET:

- displays tables compliant with the select taxonomy type and code, specified in the process of defining a set;

- has all tables active for the given taxonomy type, irrespective of whether they are prepared by the bank or not;

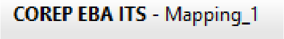
- along the given name provides information on the taxonomy type with which the created set was linked – this function is presented on Figure 9:
  - in a bookmark form:  COREP EBA ITS - Mapping_1 ,
  - in the right screen corner:  Mapping_1 COREP EBA ITS 5.1 ,

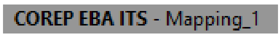
- allows for defining and saving new mapping expressions and then for their review and editing,

- displays all tables in a similar way as in case after report creation in the "Reports" module, however in this case not data but mapping expressions are entered into particular table cells; these expressions with the use of "Fetching by mappings" will allow for ***fetching external data to the reports, in accordance with the defined data source.***

"Data mapping" module can be opened simultaneously for multiple mapping sets and switching between them is performed as in the case of usual reports, by clicking the selected set.

The active set is always the one with the name displayed on the bookmark with a light background, i.e.:

 COREP EBA ITS - Mapping_1 – an active mapping set

 COREP EBA ITS - Mapping_1 – an inactive mapping set

In order to correctly **define a mapping expression** for data fetching, the user needs to:

- open a correct MAPPING EXPRESSIONS SET,
- open a table within the set for which the mapping expressions should be entered,
- set the cursor in the cell ***in which mapping expressions are to be saved,***
- depending from the initial cell type for which the mapping expression is created, ***enter an appropriate type of the formulated expression (see chapter*** 6.2 Mapping expressions***)***.

The above-described steps to perform when creating mapping expressions apply to all reporting table fields except for "Typed dimensions".

Creating mapping expressions for "Typed dimensions" is performed with the use of the "Settings" function, described in details in chapter 2.1.1.1.3 Mapping expressions set setting***.***

2.1.1.1.2 Set management

In order to proceed to **"Mapping set management"**, select:

- **"Mapping set" menu** Sets management

or

- one of the following keyboard shortcuts: **Ctrl-O** or **Ctrl-1**

or

- switch the work mode to **"List** .

Functions grouped in "Mapping set management":

- are described in details in chapter 2.1.2 List mode of this document.

2.1.1.1.3 Mapping expressions set settings

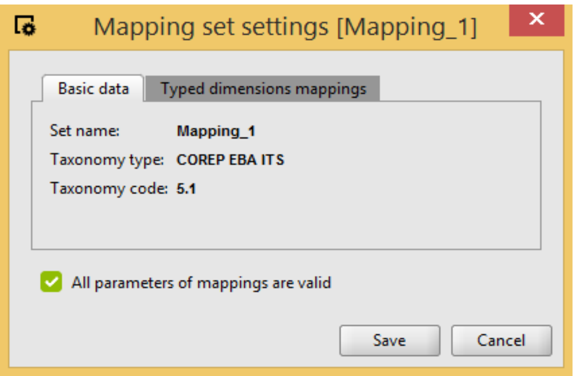
To launch "**Mapping set settings"** function, select:

- **"Mapping set" menu** Set's settings

This function:

- displays basic information about the open – active – mapping expressions set,
- for sets in which the user defines typed dimensions, allows for creation of mapping expressions for these dimensions.

When this function is launched, the window as on Figure 10 appears on screen.



Mapping set settings [Mapping_1]

Set name: Mapping_1

Taxonomy type: COREP EBA ITS

Taxonomy code: 5.1

☑ All parameters of mappings are valid

Figure 10. Mapping expressions set settings – basic data

Bookmark Basic data (see Figure 10):

- displays information about an active mapping expressions set which were entered during its creation, i.e.:
  - set name,
  - taxonomy type,
  - taxonomy code.

Bookmark Typed dimensions mappings (see Figure 11):

- in the left screen panel displays all typed dimensions possible to be defined by the bank,
- in the right screen panel allows for entering mapping expressions for the selected typed dimension.

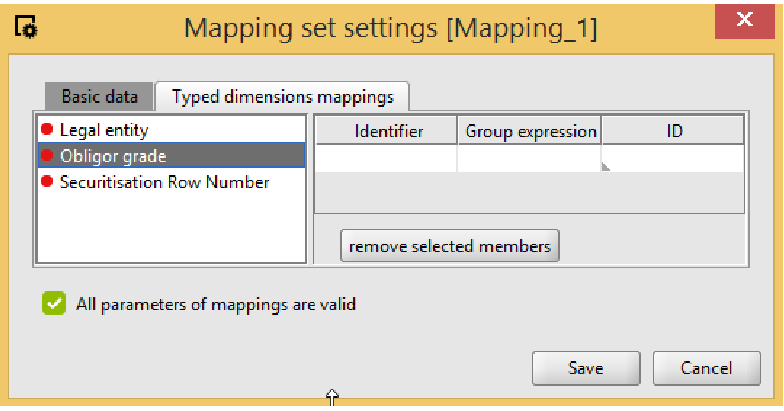

Mapping set settings [Mapping_1]
Basic data
Typed dimensions mappings
Legal entity
Obligor grade
Securitisation Row Number
Identifier
Group expression
ID
remove selected members
☑ All parameters of mappings are valid
Save
Cancel

Figure 11. Mapping expressions set settings – typed dimensions

For different typed dimensions the right window panel can display any number of columns to which appropriate mapping expressions should be entered.
The following elements are displayed in the right window panel:

- **An identifier** for which the value is set automatically by the application.
- **A grouping expression**, i.e. a field to which an expression should be entered, which refers to the typed data source and is used to separate groups within which individual values are created. For each value combination there will be new groups created, which will serve as the basis to calculate the value of a dimension and its reporting cells.

A simplified syntax of such expressions is as follows:

<source_symbol>:{ <grouping_expression>(<grouping_expression>) }

- The **remaining columns** to which expressions calculating values of particular typed dimensions during data fetching are entered. There are as many, as many elements of the given typed dimensions.

The color of the dot located along each dimension informs the user whether the mapping expressions was added to the displayed dimension in the left window panel or not, and so:

🟢 – means that the mapping expression was entered into the given dimension,

🔴 – means that the dimension does not have fully defined mapping expressions.

Mapping of typed dimensions for LE EBA ITS

For correct mapping of typed dimensions (which are "Group of connected clients" and "Individual clients") in the LE EBA ITS report, select:

- **"Mapping set" menu** Set's settings

and then

- bookmark Typed dimensions mappings

and for specific dimensions selected from the **left panel** of the window:

○ in the **right panel** of the window (see Figure 12) enter appropriate mapping expressions.

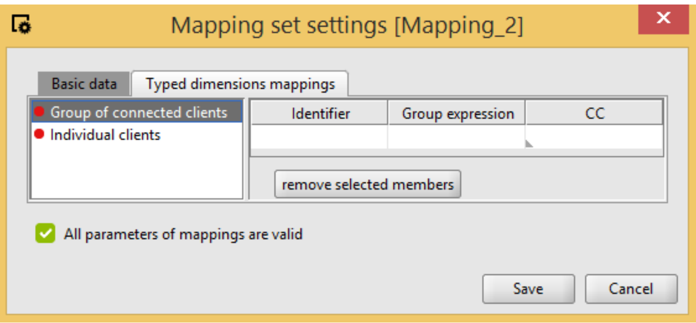

Mapping set settings [Mapping_2]
Basic data
Typed dimensions mappings
Group of connected clients
Individual clients
Identifier
Group expression
CC
remove selected members
☑ All parameters of mappings are valid
Save
Cancel

Figure 12. Mapping typed dimensions for an LE EBA ITS report – Group of connected clients

- **Grouping expression** – expression which will extract from the data source all complex groups from "Group of connected clients”.

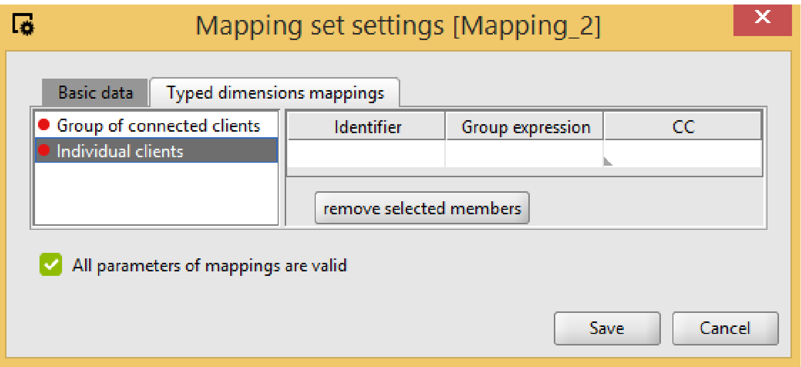
The expression defined here can have the following form:

LE:{ group($GROUP_CODE) }

- **CC** – expression calculating the "Group of the connected clients" dimension value which can have the following form:

LE:{ $GROUP_CODE }

typed dimension mappings defined in this way allow for calculation of values on individual tables by executing mapping expressions multiple times for each dimension value which results from data in source.



☑ All parameters of mappings are valid

Figure 13. Mapping typed dimensions for an LE EBA report – Individual clients

For "Individual clients" dimension (see Figure 13):

- **Grouping expression** – expression which will extract from the data source all complex groups from "Individual clients”.

The expression defined here can have the following form:

LE:{ group($ENTITY_CODE) }

- **CC** – expression calculating the " Individual clients" dimension value which can have the following form:

LE:{ case($ENTITY_CODE) }

typed dimension mappings defined in this way allow for calculation of values on individual tables by executing mapping expressions multiple times for each dimension value which results from data in source.

2.1.1.1.4 Mapping set history

To launch "**Mapping set history"** function, select:

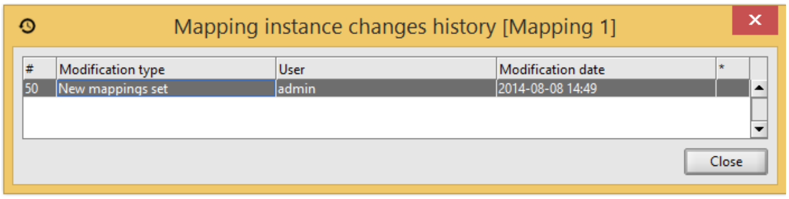
- **"Mapping set" menu** History of operations

or

- icon:

This function (see Figure 14) displays:

- history of changes implemented in the active mapping set.



Mapping instance changes history [Mapping 1]

#	Modification type	User	Modification date
50	New mappings set	admin	2014-08-08 14:49

Figure 14. History of changes implemented in the active mapping set

2.1.1.1.5 *Import mapping expressions*

To launch "**Mapping expressions import"**, select:

- **"Mapping set" menu** Import expressions

or

- icon: ,

or

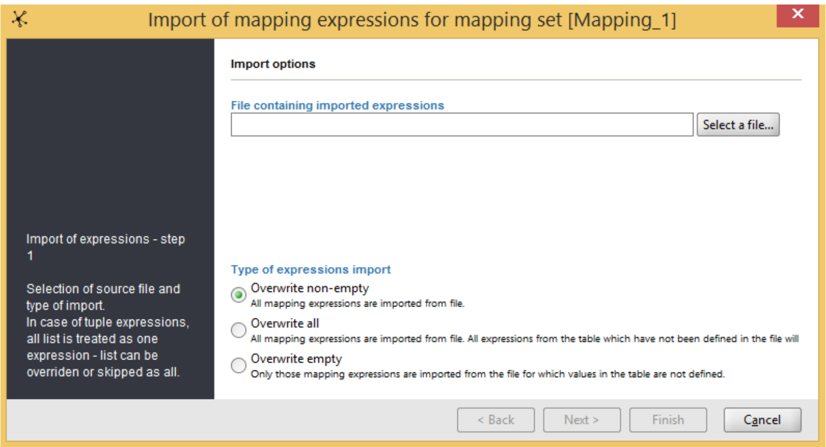
- keyboard shortcut: **Ctrl-I**.

This function:

- allows for import of previously defined mapping expressions.

Full import of mapping expressions consists each time of ***three steps***.

Step 1 – selection of the source file and import type (see Figure 15).



Import of mapping expressions for mapping set [Mapping_1]

Import of expressions - step 1

Selection of source file and type of import.
In case of tuple expressions, all list is treated as one expression - list can be overriden or skipped as all.

Import options

File containing imported expressions

Type of expressions import

● Overwrite non-empty
All mapping expressions are imported from file.

○ Overwrite all
All mapping expressions are imported from file. All expressions from the table which have not been defined in the file will

○ Overwrite empty
Only those mapping expressions are imported from the file for which values in the table are not defined.

Figure 15. Selection of the source file and expression import type

Types of expression import:

Overwrite with non-empty – will result in:

- import of all mapping expressions from an external file,
- retaining of only those previously entered mapping expressions which were not included in the imported file.

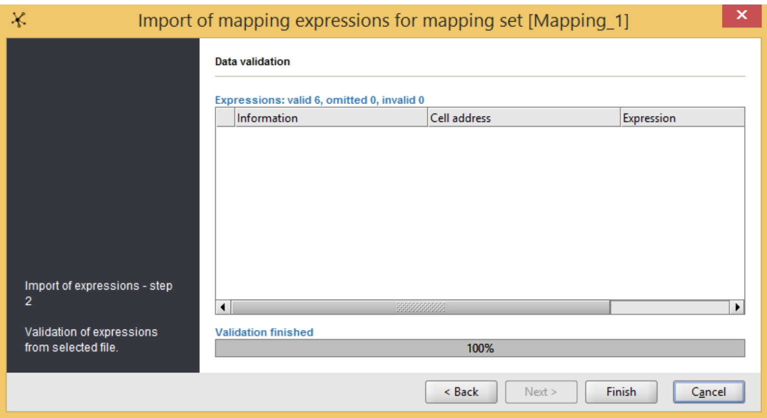
Overwrite all – will result in:

- import of all mapping expressions from an external file,
- removal of all mapping expressions which were entered before the import (applies to those reports to which expressions from an external file will be imported).

Overwrite empty – will result in:

- import from an external file of only those mapping expressions which in report tables did not have any values entered,
- retaining of all previously saved mapping expressions

Step 2 – validation of expressions included in the file to import (see Figure 16).

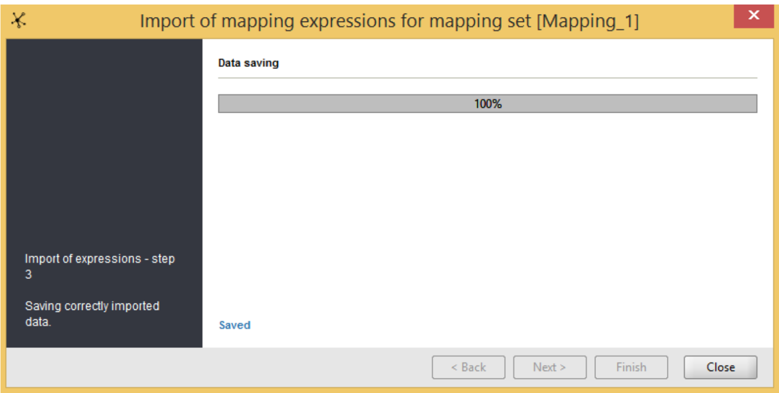

Import of mapping expressions for mapping set [Mapping_1]
Data validation
Information
Cell address
Expression
Import of expressions - step 2
Validation of expressions from selected file.
100%
< Back
Next >
Finish
Cancel

Expressions: valid 6, omitted 0, invalid 0

Validation finished

Figure 16. Validation of expressions included in the file to import

Step 3 – saving correctly imported expressions (see Figure 17).


Import of mapping expressions for mapping set [Mapping_1]
Data saving
100%
Import of expressions - step 3
Saving correctly imported data.
< Back
Next >
Finish
Close

Saved

Figure 17. Saving correctly imported expressions

2.1.1.1.6 *Export mapping expressions*

To launch "**Mapping expressions export"**, select:

- **"Mapping set" menu** Export expressions

or

- icon:

or

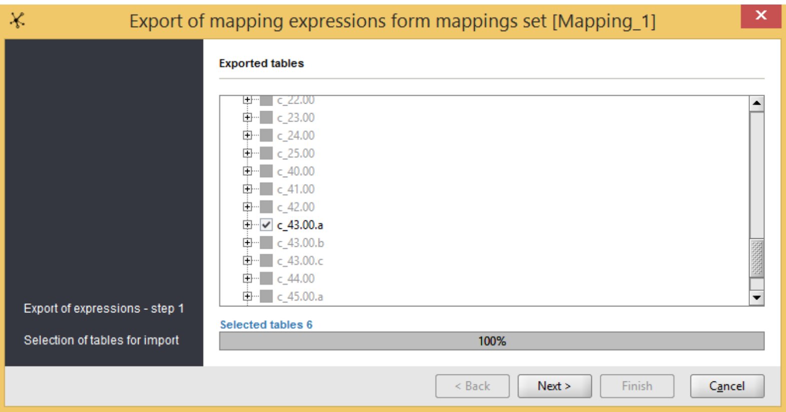
- keyboard shortcut: **Ctrl-E**

This function:

- allows for export of mapping expressions defined in an active mapping set,
- enables performance of mapping export from all or only certain tables,
- saves exported data in XML or XLSX format.

Full mapping expression export process is performed in **three steps**.

Step 1 – selecting tables to export (see Figure 18).



Export of mapping expressions form mappings set [Mapping_1]

Exported tables

□ c_22.00
□ c_23.00
□ c_24.00
□ c_25.00
□ c_40.00
□ c_41.00
□ c_42.00
☑ c_43.00.a
□ c_43.00.b
□ c_43.00.c
□ c_44.00
□ c_45.00.a

Selected tables 6

100%

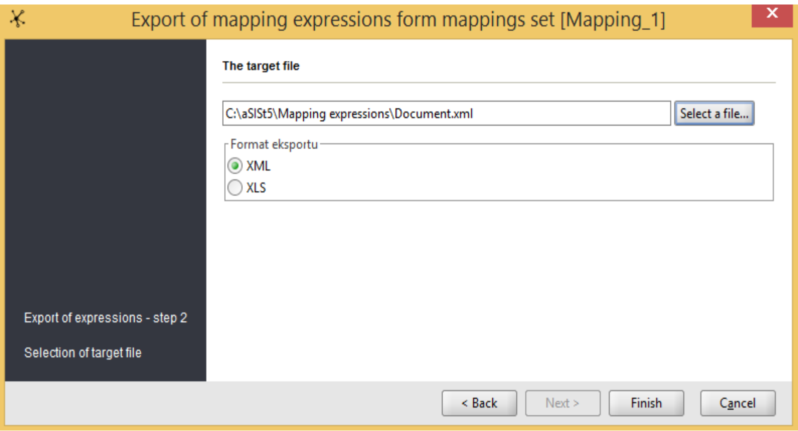
Export of expressions - step 1

Selection of tables for import

Figure 18. Selecting tables to export

Only those previously defined expressions with tables marked on the displayed screen will be exported.

Step 2 – selecting the target file with exported expressions (see Figure 19).


Export of mapping expressions form mappings set [Mapping_1]

The target file

C:\aSISt5\Mapping expressions\Document.xml

Format eksportu
● XML
○ XLS

Export of expressions - step 2

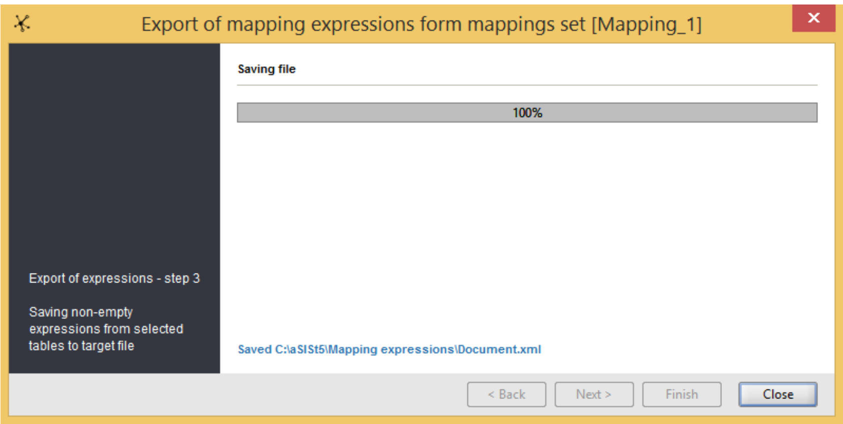
Selection of target file

Figure 19. Selecting the target file for saving exported expressions

In this process it is necessary to:

- select a path to save the exported mapping expressions,
- provide a name for the file created.

Step 3 – saving non-empty expressions from selected tables (see Figure 20).


Export of mapping expressions form mappings set [Mapping_1]

Saving file

100%

Export of expressions - step 3

Saving non-empty expressions from selected tables to target file

Saved C:\aSISt5\Mapping expressions\Document.xml

Figure 20. Saving exported non-empty expressions from selected tables

After the exported expressions are saved, exit from this function can be done by selecting the Close button.

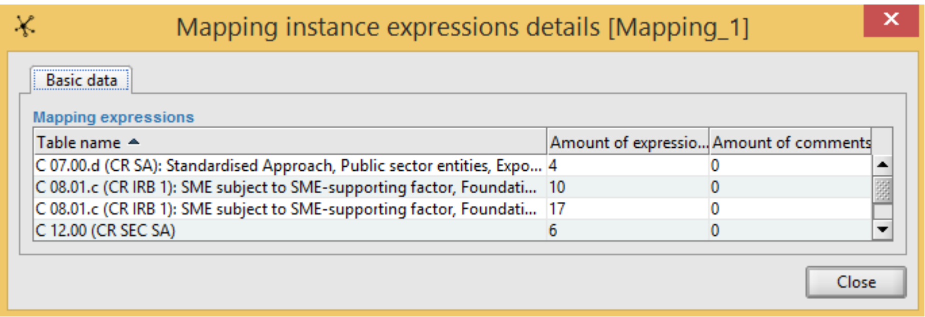
2.1.1.1.7 Information about expressions

To launch "**Information about mapping set expressions"** function, select:

- **"Mapping set" menu** Expressions information

This function (see Figure 21) displays:

- list of tables in which mapping expressions were defined,
- number of expressions and commentaries for those tables.



Mapping instance expressions details [Mapping_1]

Basic data

Mapping expressions

Table name	Amount of expressio...	Amount of comments
C 07.00.d (CR SA): Standardised Approach, Public sector entities, Expo...	4	0
C 08.01.c (CR IRB 1): SME subject to SME-supporting factor, Foundati...	10	0
C 08.01.c (CR IRB 1): SME subject to SME-supporting factor, Foundati...	17	0
C 12.00 (CR SEC SA)	6	0

Figure 21. Mapping set details

2.1.1.1.8 *Remove mapping expressions*

To launch "**Remove mapping expressions"**, select:

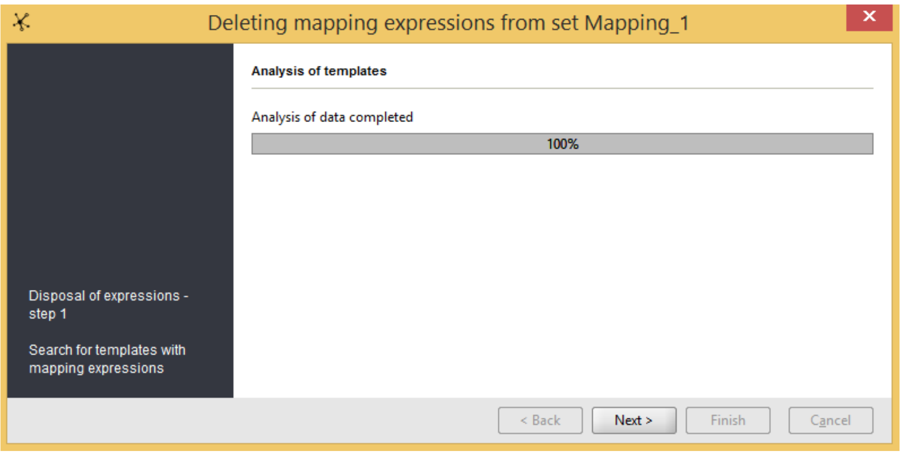
- **"Mapping set" menu** Delete expressions

This function:

- allows for removal of all expressions from an active mapping set.

Full mapping expression removal process is performed in **three steps**.

Step 1 – search for forms filled in with mapping expressions (see Figure 22).



Deleting mapping expressions from set Mapping_1

Analysis of templates

Analysis of data completed

100%

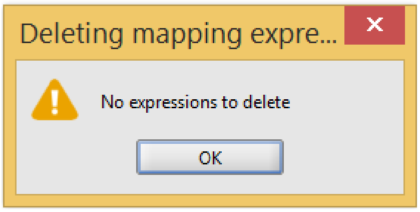
Disposal of expressions - step 1

Search for templates with mapping expressions

Figure 22. Form analysis in mapping expressions removal

The search through all forms in an active mapping set is executed in order to verify if it contains forms filled in with mapping expressions.

In case there are no such forms, the user is informed about it by a message as on Figure 23.

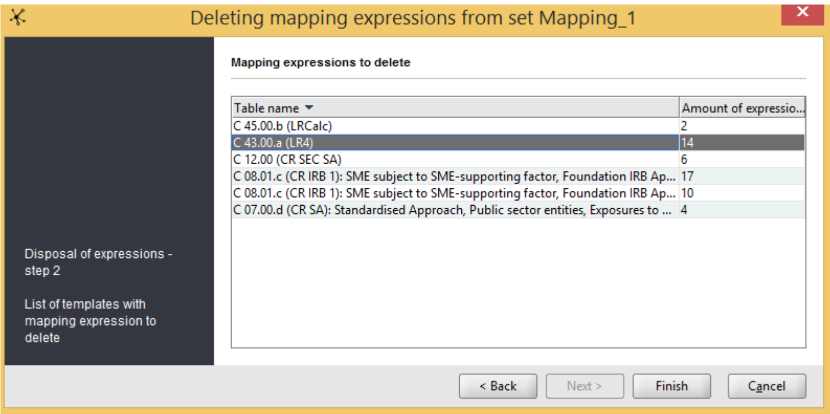


Deleting mapping expre...

No expressions to delete

Figure 23. Message about the lack of mapping expressions

Step 2 – displaying forms containing mapping expressions to remove (see Figure 24).



Deleting mapping expressions from set Mapping_1

Disposal of expressions - step 2

List of templates with mapping expression to delete

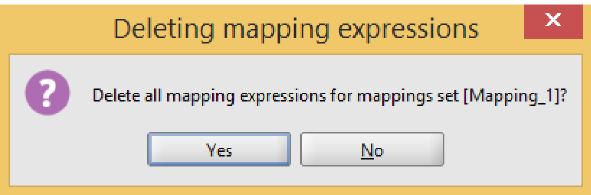
Mapping expressions to delete

Table name	Amount of expressio...
C 45.00.b (LRCalc)	2
C 43.00.a (LR4)	14
C 12.00 (CR SEC SA)	6
C 08.01.c (CR IRB 1): SME subject to SME-supporting factor, Foundation IRB Ap...	17
C 08.01.c (CR IRB 1): SME subject to SME-supporting factor, Foundation IRB Ap...	10
C 07.00.d (CR SA): Standardised Approach, Public sector entities, Exposures to ...	4

Figure 24. Displaying forms with mapping expressions to remove

List of all forms which contain mapping expressions to be removed if the removal process is continued is displayed on the screen.

Selecting the Cancel button will cause a display of an additional message, as on Figure 25.



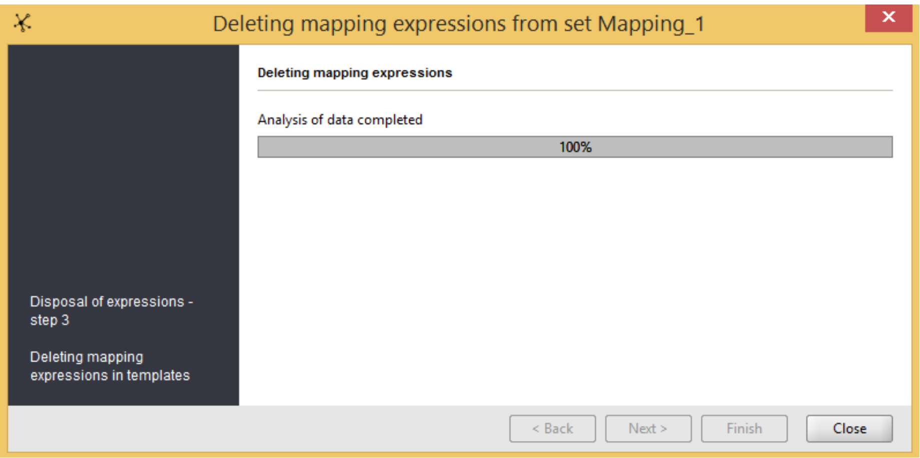
Deleting mapping expressions

Delete all mapping expressions for mappings set [Mapping_1]?

Figure 25. Additional message about the mapping expressions removal

When the message is approved all mapping expressions are removed from the previously displayed forms.

Step 3 – correct removal of mapping expressions (see Figure 26).


Deleting mapping expressions from set Mapping_1

Deleting mapping expressions

Analysis of data completed

100%

Disposal of expressions - step 3

Deleting mapping expressions in templates

Figure 26. Correct removal of mapping expressions

Selection of Close button finishes the mapping expressions removal process and the active mapping set will not contain any previously defined mappings.

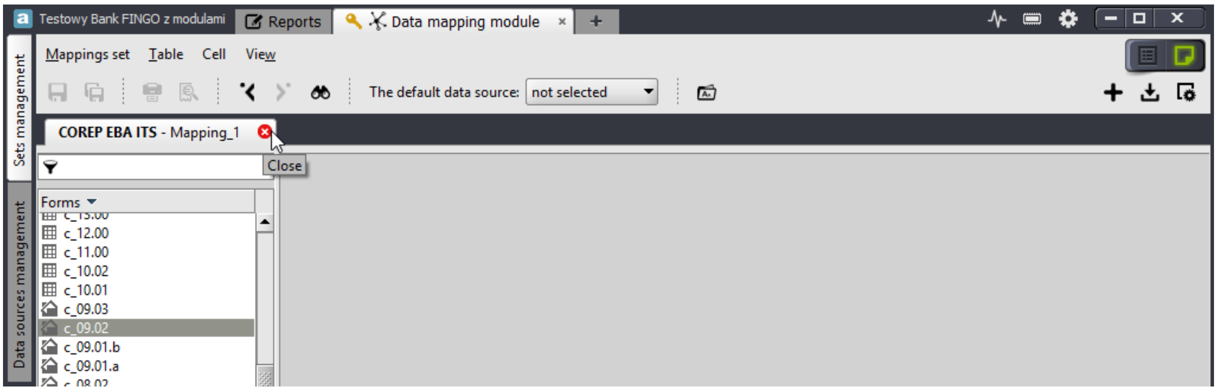
2.1.1.1.9 *Close mapping set*

To perform "**Close mapping set**", select:

- **"Mapping set" menu** Close
- and one of the following options Close active set / Close other sets / Close all sets

or

- click: ✕ of the selected mapping set bookmark (when hovered over with the mouse cursor it changes color to red ✖ – see Figure 27).



The default data source: not selected

Figure 27. Closing a mapping set

This function:

- allows for closing of one or more mapping sets by selecting an appropriate option,
- can be selected only from the "Documents" lev .

The same functions are available and can be used by right-clicking on the bookmark of one of the open mapping sets.

2.1.1.2 Table

"Table" menu – groups functions (see Figure 28) which apply to actions performed on tables of an active mapping expressions set, including:

- saving changes implemented in the active table and all tables,
- quick search for the required table,
- quick switching between tables,
- removal of an active table from all mapping expressions,
- compressing expressions in the table,
- setting parameters of the print out,
- printing tables with entered mapping expressions,
- generating mapping expressions,
- export of defined mapping expressions to an Excel file.

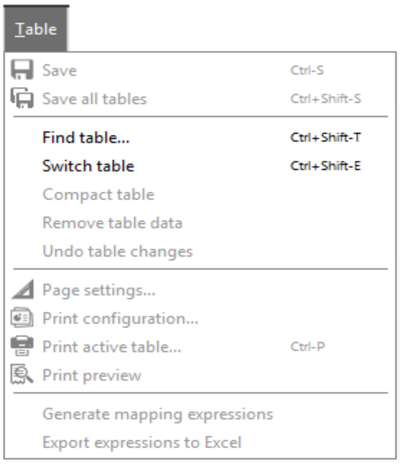


Table
Save Ctrl-S
Save all tables Ctrl+Shift-S
Find table... Ctrl+Shift-T
Switch table Ctrl+Shift-E
Compact table
Remove table data
Undo table changes
Page settings...
Print configuration...
Print active table... Ctrl-P
Print preview
Generate mapping expressions
Export expressions to Excel

Figure 28. "Table" menu in "Documents" mode

2.1.1.1.10 Generate mapping expressions

To perform "**Generate mapping expressions"**, select:

- **"Table" menu** Generate mapping expressions

This function:

- allows for automatic generation of mapping expression by the application itself without the necessity to define them manually,
- can be used only in a cell data source,
- always refers to one open table.

When this function is launched, a window as on Figure 29 is displayed on screen.

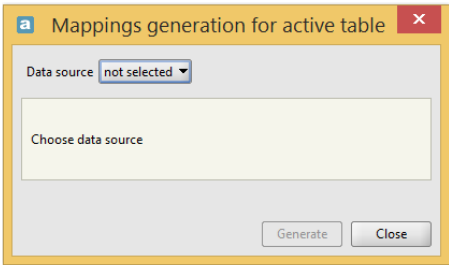
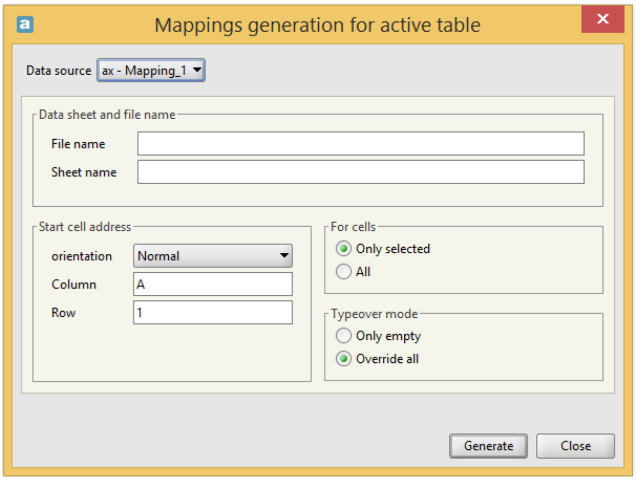


Figure 29. Data source selection before mapping expressions are generated

It is necessary to specify a cell data source on the basis of which the system can perform mapping expressions generation – only in this case the function will be activated (see Figure 30).



Data source: ax - Mapping_1

Data sheet and file name
File name:
Sheet name:

Start cell address
orientation: Normal
Column: A
Row: 1

For cells
● Only selected
○ All

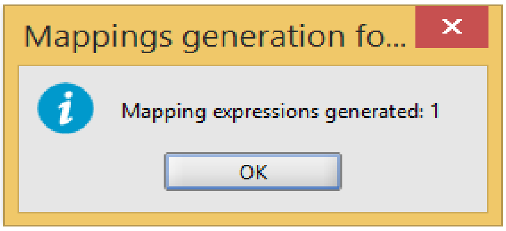
Typeover mode
○ Only empty
● Override all

Figure 30. Mapping generation for an active table

Prior to mapping expressions generation it is necessary to specify conditions which should be met for the generated expressions to meet our expectations, that is:

- provide a file/sheet name,
- specify the initial cell location from which mapping expressions defining should take place,
- select cells for which mappings (only marked / all) are to be generated,
- specify the way of entering prepared mappings (only empty / overwrite all).

Approval of selected choices by clicking the Generate button will cause generation and entering to specific (previously selected) cells appropriate mapping expressions about which the user is informed with a message as on Figure 31.

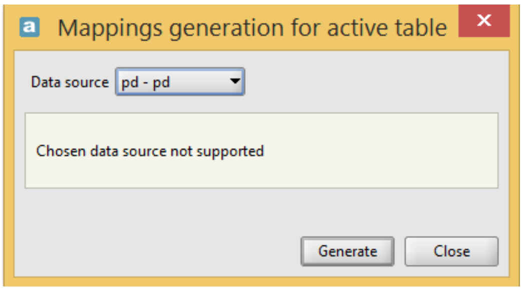


Mapping expressions generated: 1

Figure 31. Message about generation of mapping expressions

Selecting the OK button finishes the function execution.

In case data source specified are different than cell Excel form, the function will not be able to be executed and a message, as on Figure 32 will appear.



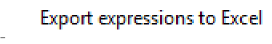
Data source pd - pd

Chosen data source not supported

Figure 32. Selection of other than cell data source while executing mapping expressions generation function

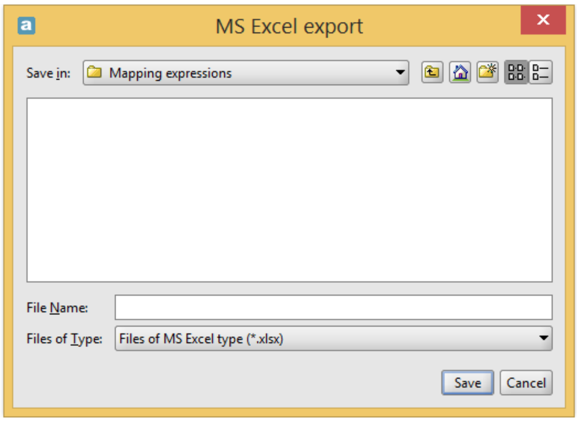
2.1.1.1.11 *Exporting expressions to Excel*

To perform "**Export expressions to Excel**", select:

- **"Table" menu** 

This function:

- allows for exporting defined mapping expressions in an Excel form,
- always applies to mapping expressions defined from an open table of an active mapping set,
- in order to be performed correctly requires (see Figure 33):
  - destination path,
  - file name.



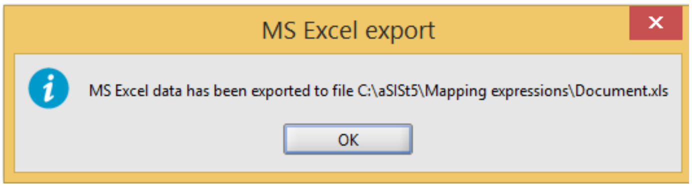
Save in: Mapping expressions

File Name:

Files of Type: Files of MS Excel type (*.xlsx)

Figure 33. Mapping expressions export to Excel

The user is informed about the correct function execution with a message as on Figure 34.



MS Excel data has been exported to file C:\aSISt5\Mapping expressions\Document.xls

Figure 34. Message about saving cell in Excel format

2.1.1.3 Cell

"**Cell**" menu – groups functions (see Figure 36) linked with actions performed on single table cells, i.e.:

- calculate highlighted expression,
- find a cell,
- find/replace value,
- move to previous/next highlighted cell,
- cell mapping history,
- cell characteristics.

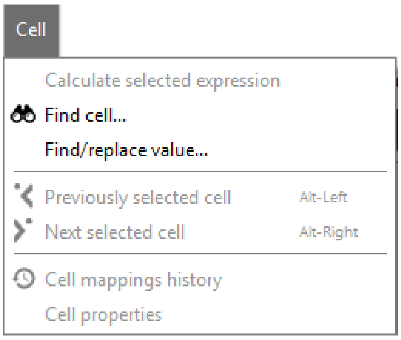

Cell
Calculate selected expression
Find cell...
Find/replace value...
Previously selected cell Alt-Left
Next selected cell Alt-Right
Cell mappings history
Cell properties

Figure 35. "Cell" menu in "Documents" mode

2.1.1.4 View

"View" menu groups (see Figure 36) functions connected with number and way of data display on the screen, i.e. functions which allow to:

- move to set list,
- make changes in the way table descriptions are displayed,
- enable/disable different information displayed on screen: list of errors, details of defined expressions, XBRL details or search results;
- enter comments into cells,
- enter changes in the amount of information displayed on screen, i.e. enable/disable: form list, editing panel, tools panel, status panel, full screen mode;
- display cells with calculations,
- wrap long texts,
- turn list tables,
- revert to default setting of an active table.

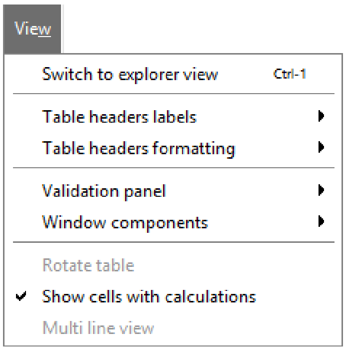
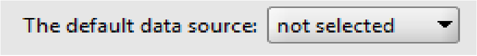


Figure 36. "View" menu in "Documents" mode

2.1.1.5 Default data source

To define a **"Default data source"** proceed as follows:

- expand the list of available (defined) data sources, i.e.:



The default data source: not selected

- ***select one of the previously defined data sources as a default source.***

This function:

- can be used only when the required data source was previously defined,
- allows for simplified building of mapping expressions.

For example:

The defined expression in which a data source identifier was provided explicitly ***(Source_1):***

Source_1:{ sum($deposits, $surname = Johnson) }

If the required data source is selected as a default one, then the expression can have the following form:

{ sum($deposits, $surname = Johnson) }

and the data source symbol will be included automatically in the expression.

2.1.2 List mode

"List" mode for which the document switch view is as follow is a mode in which:

- a list of all created mapping expressions sets is displayed,
- functions possible to execute (see Figure 37) are grouped in a menu:
  - Mapping set
  - View

Switching from the "Documents" mode to the **"Mapping sets list" mode** is possible after selecting:

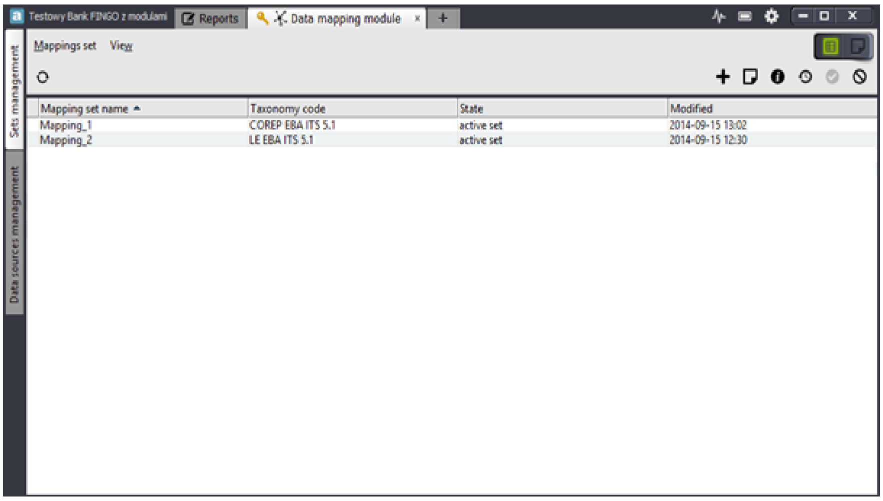
- **"Mapping set" menu** Sets management

or

- one of the following keyboard shortcuts: **Ctrl-O** or **Ctrl-1**

or

- the "Documents view switch", i.e .



Mapping set name	Taxonomy code	State	Modified
Mapping_1	COREP EBA ITS 5.1	active set	2014-09-15 13:02
Mapping_2	LE EBA ITS 5.1	active set	2014-09-15 12:30

Figure 37. Mapping sets management – list of available sets of mapping expressions

On the displayed list the following elements are shown in individual columns:

- mapping set name,
- code of the taxonomy to which the set belongs,
- mapping set status (**inactive sets** are displayed in a **brighter font**),
- date of the last set modification.

2.1.2.1 Mapping set

"Mapping set" menu (see Figure 38) groups basic **functions connected with mapping sets,** i.e.:

- creating a new mapping expressions set,
- opening previously created mapping set,
- reviewing mapping set details,
- changing the mapping set status, i.e. deactivating/activating mapping sets.

Except for the "New set" function all remaining functions:

- apply to a specific mapping set selected from the displayed list,
- become active when the required set is highlighted.

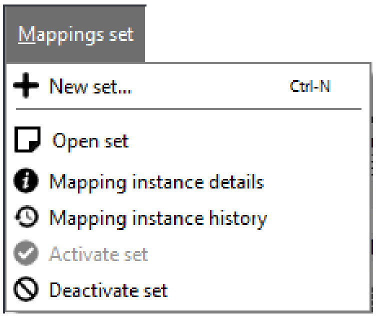


Figure 38. "Mapping set" menu functions – "Mapping sets list" mode

2.1.2.1.1 *New mapping expressions set*

To create **"New mapping expressions set"**, select:

- **"Mapping set" menu** + New set...

or

- icon: +,

or

- keyboard shortcut: **Ctrl-N**

This function:

- is available in both work modes, i.e.:
  - in "Documents" mode - ▤ ◪ ,
  - in "List" mode – ▤ ◪ ,
- is described in details in chapter 2.1.1.1.1 New mapping expressions set.

2.1.2.1.2 *Open a mapping expressions set*

To open a previously created mapping set proceed as follows:

in the **"Mapping sets list"** mode use the **"Open mapping set"** function by:

- **marking (highlighting) the set which should be opened on the list,**

and:

- selecting **"Mapping set" menu** Open set

or

- clicking icon

or

- **double-clicking with left mouse button on the selected set.**

This function:

- is available only in the "List" mode ,
- is activated only when the set to be opened is marked (highlighted) on the list,
- opens the selected set and automatically launches the "Documents" mode in the application .

All previously opened sets will remain open and visible in the "Documents" mode but will not be active and their bookmarks will be displayed with a darker background.

2.1.2.1.3 Mapping set details

To launch "**Mapping set details"** function, select:

- **"Mapping set" menu** Mapping instance details

or

- icon .

This function:

- provides information about mapping expressions set setting as on Figure 10 (see chapter 2.1.1.1.3 Mapping expressions set settings), however there is no possibility here to make any changes,
- is activated only when the mapping set with details to be reviewed is marked (highlighted) on the set list.

2.1.2.1.4 Mapping set history

To launch "**Mapping set history"** function, select:

- **"Mapping set" menu** Mapping instance history

or

- icon: .

This function (see example Figure 12) displays:

- history of changes implemented in the active mapping set,
- is available in both modes, i.e. "List" and "Documents" ;
- in the "List" mode is activated only when the set to be reviewed is marked (highlighted) on the set list.

2.1.2.1.5 Activate set

To launch "**Activate mapping set"** function, select:

- **"Mapping set" menu** Activate set

or

- icon ✓.

This function:

- is available only in the "List" mode ,
- is activated only when the set with an "inactive" status (i.e. which was earlier deactivated) is marked (highlighted) on the list,
- changes the set status to "active".

The mapping set with an **"active"** status:

- can be opened and edited,
- mapping expressions formulated in it can be exported (see chapter 2.1.2.1.6 Export mapping expressions).
- can have other mapping expressions imported to it (see chapter 2.1.2.1.7 Information about expressions),
- can be used to fetch external data to the reports (see chapter 5 Fetching by mappings).

2.1.2.1.6 Deactivate set

To launch "**Deactivate mapping set"** function, select:

- **"Mapping set" menu** ⦸ Deactivate set

or

- Icon ⦸.

This function:

- is available only in the "List" mode ▤ ◪ ,
- is activated only when the set with the "active" status is marked (highlighted) on the list,
- changes the set status to "inactive".

The mapping set with an **"inactive"** status:

- can be opened as a read-only,
- cannot be edited,
- mapping expressions formulated in it can be exported, but
- other mapping expressions cannot be imported to it,
- is invisible for the "Fetching by mappings" function and for this reasons cannot be used to fetch external data to reports,
- can be activated at any time.

2.1.2.2 View

"View" menu (see Figure 39), enables the function linked with:

- switching work into the "Documents" mode.

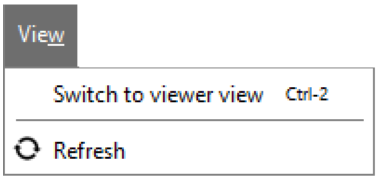


Figure 39. "View" menu in the "Mapping expressions list" mode

2.1.2.1.7 Switching to "Documents" list

Switching from the "Mapping set list" mode to the **"Documents" mode** is possible after selecting:

- **"View" menu** Switch to viewer view

or

- keyboard shortcut: **Ctrl-2**

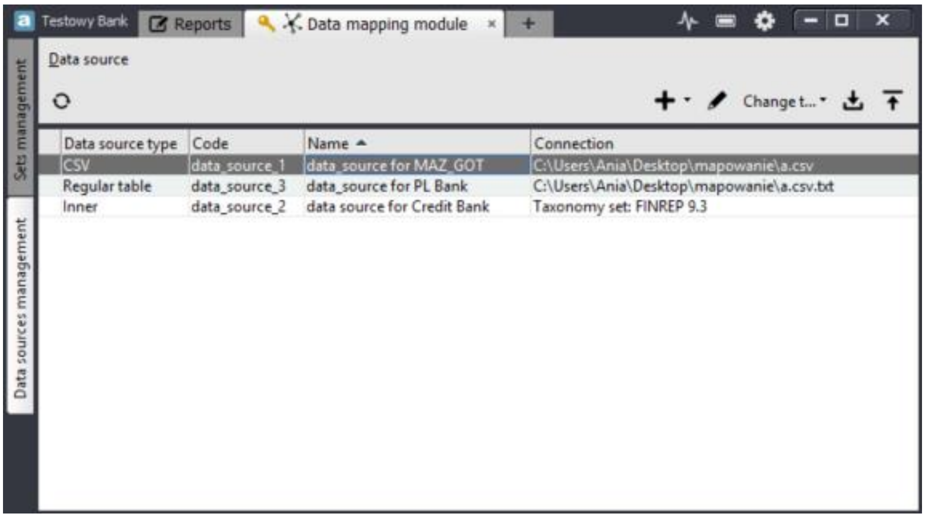
or

- the "Documents view switch", i.e.. .

2.2 Data source management

Activation of the **"Data mapping"** module and selection of the **"Data source management"** side bookmark (see Figure 40) allows for:

- **providing information about defined data sources** by:
  - displaying a list of defined sources,
- **defining new and managing existing data sources** by:
  - entering changes to existing sources,
  - exporting and importing data sources,
  - reviewing history of changes implemented in data source,
- **quick calculation of defined mapping expressions on the basis of the data source specified.**



Data source type	Code	Name	Connection
CSV	data_source_1	data_source for MAZ_GOT	C:\Users\Ania\Desktop\mapowanie\a.csv
Regular table	data_source_3	data_source for PL Bank	C:\Users\Ania\Desktop\mapowanie\a.csv.txt
Inner	data_source_2	data source for Credit Bank	Taxonomy set: FINREP 9.3

Figure 40. Data source management

All functions are made available here:

- in the **"Data source"** menu,
- in the form of functional icons.

2.2.1 Data sources

"Data source" menu (see Figure 41) groups functions linked with:

- data source management,
- quick calculation of any mapping expression with the use of the available data source.

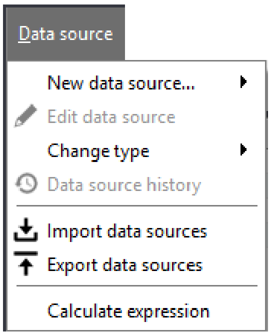


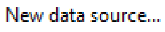
Figure 41. "Data source" menu in Data source management

Data sources are:

- global for the entire application,
- shared by the mapping expressions which use them.

2.2.1.1 New data source

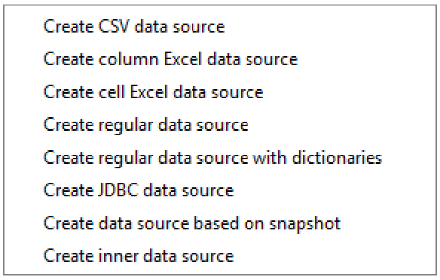
To define a **"New data source"** select in the **"Data source management"** bookmark:

- **"Data source" menu** 

or

- Icon +,

and one of the following options:


Create CSV data source
Create column Excel data source
Create cell Excel data source
Create regular data source
Create regular data source with dictionaries
Create JDBC data source
Create data source based on snapshot
Create inner data source

This function allows for defining a new data source:

- of a type compliant with the selection made at the function launch,
- not connected with any taxonomy or report – global for the entire application,

- shared by the mapping expressions which use them.

When the function is launched one of the screens presented on the Figures 42 – 48 is displayed.

For different data source types a different amount of data to define is required; data are grouped in bookmarks:

- basic data,
- parameters,
- virtual columns,
- formats,
- dictionaries.

2.2.1.1.1 *Basic data of the new data source*

In the Basic data bookmark (see Figure 42 - 48) the user needs to provide:

- **an identifier** – a unique code of the defined data source, used in the expressions built – for **all data source types**,
- **long name** – a full name of the defined name data source – for **all data source types**,
- **snapshot names** – for the **snapshot data source** (see Figure 48),
- **SQL queries** – for the **snapshot data source** (see Figure 48),
- **data path type**, displayed in the form of:
  - file,
  - catalog,
  - zip archive.

*There is a possibility to choose **all file data sources (except for the "Table with dictionaries" and "snapshot data source").***

- **data access paths** – *for all **file data sources (except for "Snapshot data source),***
- **filter** – facilitating additional filtering of files in case of catalog or ZIP archive selection

*for **all file data source,***

- **data separator in the form of:**
  - ;
  - |
  - <TAB>

*for **CSV files*** (see Figure 42):

- **coding type:**

- ISO-8859-1 (default),
- ISO-8859-2,
- Windows-1250,
- UTF-8,
- UTF-16,
- US-ASCII,
- other

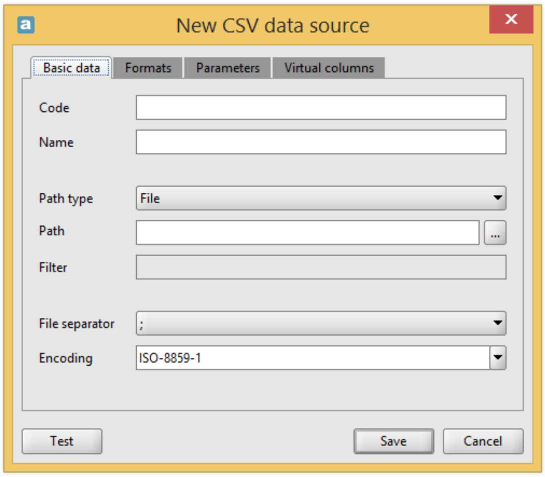
*for **CSV and Regular data source***

- **spreadsheet number** or **spreadsheet name** – for **column Excel** and **cell Excel** files (see Figures 43, 44),
- **regular data source table type:**
  - regular header,
  - regular table or
  - fixed columns

for files type: ***Regular*** (see Figure 45):

- **basic data source** – indicating other existing data source

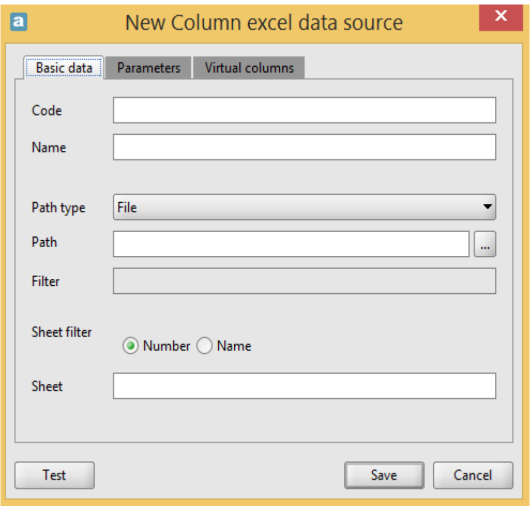
*for **"Table with dictionaries" data type*** (see Figure 46).



New CSV data source

Code:
Name:
Path type: File
Path:
Filter:
File separator: ;
Encoding: ISO-8859-1

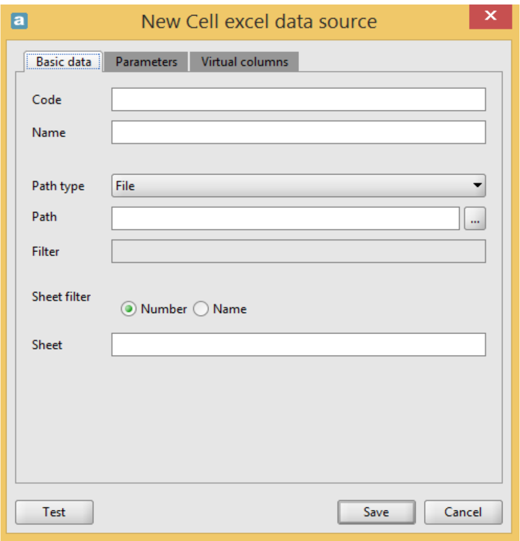
Figure 42. Defining a new source for CSV data source type



Path type: File

Sheet filter: ● Number ○ Name

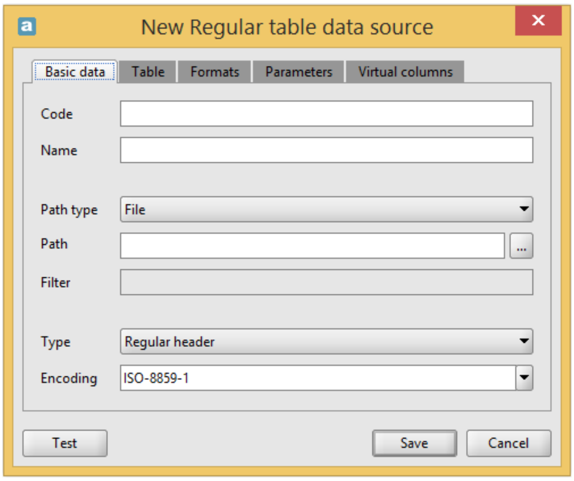
Figure 43. Defining a new source for column Excel data source type



Path type: File

Sheet filter: ● Number ○ Name

Figure 44. Defining a new source for cell Excel data source type

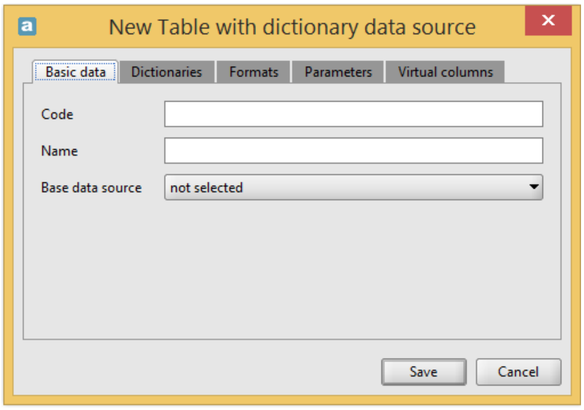


Path type: File

Type: Regular header

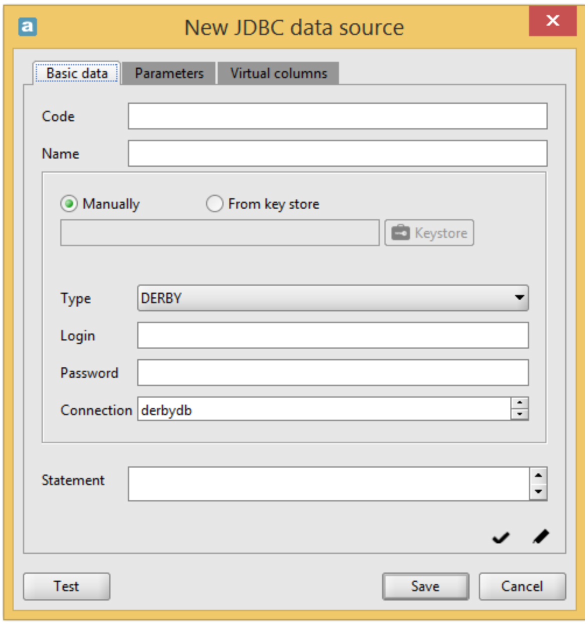
Encoding: ISO-8859-1

Figure 45. Defining a new source for Regular data source type



Base data source: not selected

Figure 46. Defining a new source for Table with dictionaries data source type



● Manually ○ From key store

Type: DERBY

Connection: derbydb

Figure 47. Defining a new source for IDBC data source type

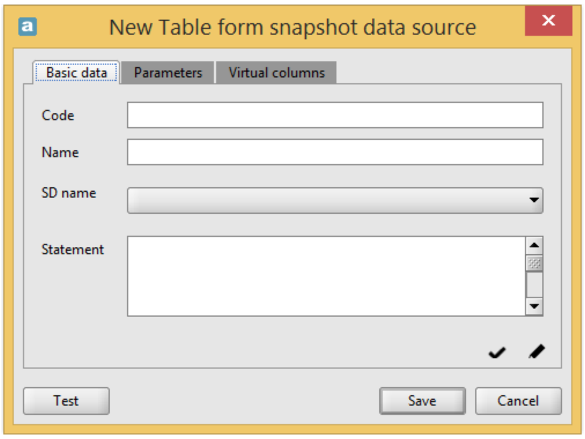
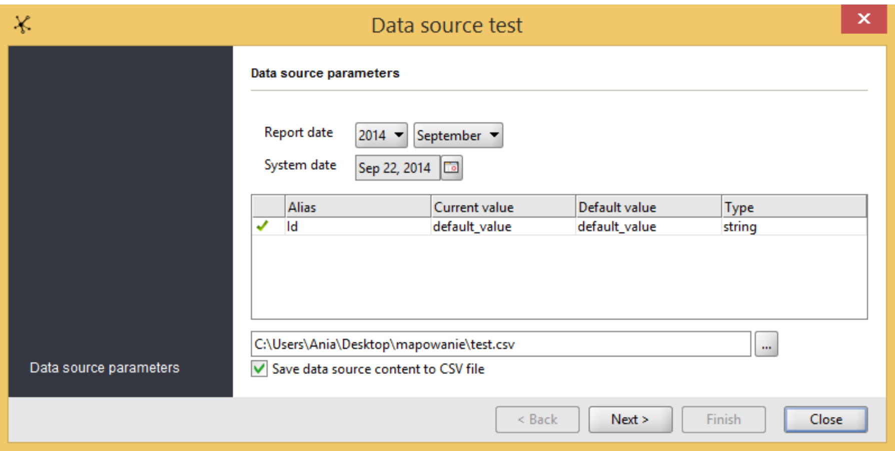


Figure 48. Defining a new source for "Snapshot data source" data source type

After required data are entered for all data sources except the "Table with dictionaries" it is possible to verify correctness of the defined settings and display a **fragment** of test data.

Such correctness verification of setting is possible to perform with the Test button (see Figures 42, 43, 44, 45, 46, 48).



Data source test

Data source parameters

Report date 2014 September

System date Sep 22, 2014

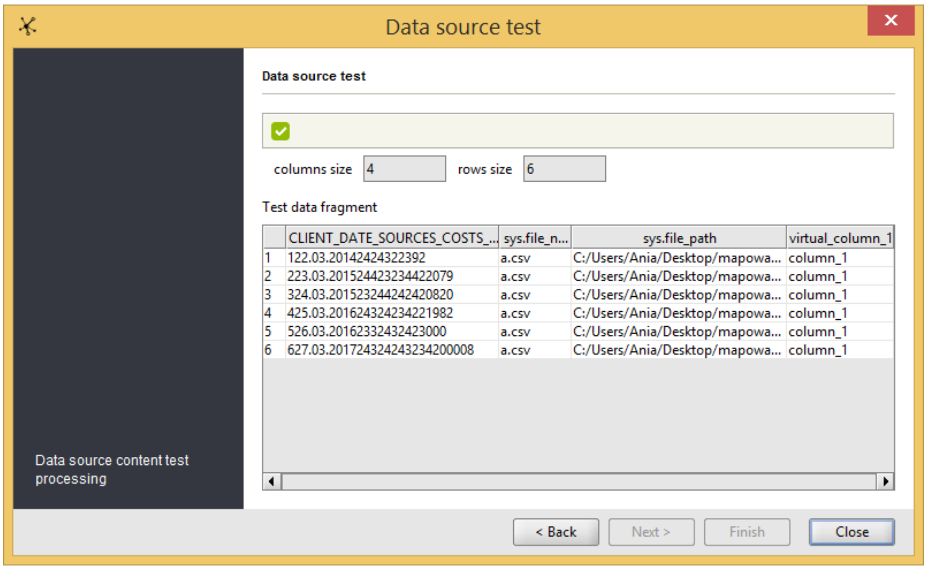
	Alias	Current value	Default value	Type
✓	Id	default_value	default_value	string

C:\Users\Ania\Desktop\mapowanie\test.csv

☑ Save data source content to CSV file

Figure 49. Data source test in the "mappings module"

After the Next > button is selected, the entire data source table is saved in the CSV format. This way a full preview of the table with virtual and system columns will be obtained. This function facilitates search for errors in the definition or incorrect values in the data source.



Data source test

Data source test

✓

columns size 4 rows size 6

Test data fragment

	CLIENT_DATE_SOURCES_COSTS_...	sys.file_n...	sys.file_path	virtual_column_1
1	122.03.20142424322392	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1
2	223.03.201524423234422079	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1
3	324.03.201523244242420820	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1
4	425.03.201624324234221982	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1
5	526.03.20162332432423000	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1
6	627.03.201724324243234200008	a.csv	C:/Users/Ania/Desktop/mapowa...	column_1

Figure 50. Data source test results in the "mappings module"

Above the displayed data fragment, information about the total number of rows and columns of the tested data source is displayed. After selecting Close , window panel will close.

Tabular data sources.

All supported data sources require the source definition to describe how to obtain a correct table from the source data.

Correctly prepared tabular data source is a table with:

- a header with unique column names,
- rows with data having the same number of columns.

Column names need to comply with the following format:
[a-zA-Z][a-zA-Z0-9_]*

It means that all characters in column names not matching this format (also white characters) will be replaced with the following character: '_'.
White characters in the beginning and the end of the column name are removed. For example the name:
'initial bal.'
in the process of source download will be replaced with:
'initial_bal_'

Depending on the data source there is a possibility of having additional columns filled in by the system. Such column are called system columns and their name begins with the prefix 'sys'.
System column referring to 'value' looks as follows:
$sys.value

CSV, column Excel, cell Excel and regular data source types are tabular source files. It means that data downloaded from the table defined in the file (or files) correspond to the data source type.
The path specifies therefore a file, a catalog or a ZIP archive with files in an appropriate format.

In case a catalog or a ZIP archive is specified instead of one file, it is possible to load multiple files in the same format.
It means that multiple files will be joined and treated as one table with rows from all files.
Such operation requires all specified files to contain a table with the same header.

To limit names of the files loaded within a catalog or a ZIP archive, a **filter** needs to be defined.

Filters use a simple mechanism of regular expressions:

- * denotes any text,
- ? denotes one character,
- [<characters>] denotes one character from the <characters> set.

For example:
[abc]*.csv

Denotes all files beginning with 'a', 'b' or 'c' and ending with '.csv'.

A default filter is empty, i.e. identical with the '*' filter.

The remaining attributes of the data source definition depend on its type.

CSV (Comma Separated Values) is a file source in which:

- the path specifies a text file or files containing values separated with the selected separator,
- column names are located in the first file row, the remaining rows contain data, individual fields are separated with a selected separator (';' or ',' or 'I' or <TAB>); in case an incorrect separator is specified it can happen that all fields will be joined into one column,
- file coding allows for correct identification of national characters.

'Column Excel' and 'cell Excel' types are sources in which:

- the path specifies a file or files compliant with Excel 97-2003 format,
- spreadsheet specifies number and name of the spreadsheet:
    - in case the number is specified – value has to be a positive number of an existing spreadsheet in the file (spreadsheets are numbered beginning with 1),
    - in case the name is specified – value is a filter for the spreadsheet name which means that it might specify more than one spreadsheet; filter for the spreadsheet is construed in the same way as the filter for the file name (described above),
- dates occur in cells of a textual type.

Additionally, in **'column Excel'**:

- each column has a unique header, the header is located in the first row of the selected spreadsheet,
- the table begins in the first column and the first row of the selected spreadsheet (A1 cell),
- all remaining rows contain data only in columns with a header,
- spreadsheet <u>does not contain</u> other data apart from those in the table.

Additionally, in **'cell Excel'**:

- specified spreadsheets contain any data,
- data are laid out in the table with the following header:

sys.address	sys.column	sys.row	sys.value
A1	A	1	<value>

- classical expression language in the tables and simplified notation are supported, which facilitates referring to a value of exactly one spreadsheet cell, for example:

@A1

"Regular data source" type allows for specifying the manner in which a table was included in the text file:

- the path specifies text file or files in which a table with data is located,
- file coding allows for correct identification of national characters,
- table type specifies content of the Table bookmark and at the same type – finding the table in the file.

"Tabular data source with dictionaries" type allows for joining multiple previously defined data sources:

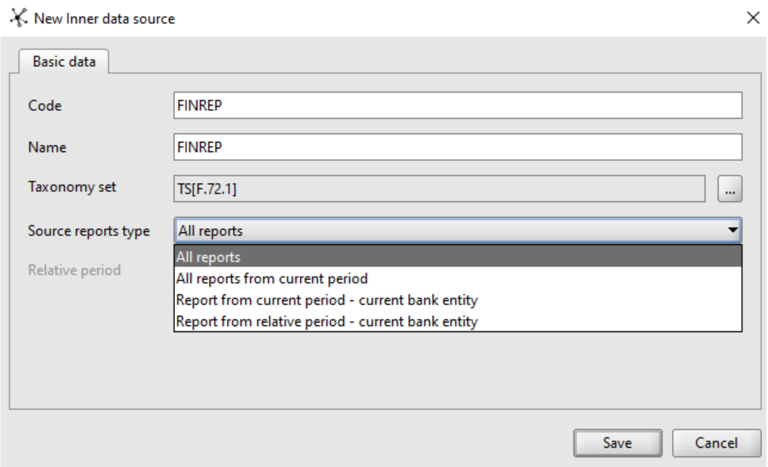
- includes reference to the main data source,
- table in this source has the same form as in the basic data source,
- bookmark Dictionaries enables building dictionaries on the basis of existing data sources.

"Snapshot data source" type allows for joining, with the use of the created SQL query, many tables derived from the snapshot or further steps of ETL transformations, defined in snapshot form in the "Snapshot warehouse":

- contains reference to snapshots from "Snapshot warehouse",
- table in this source can have an entirely different form than tables in the basic data source,
- SQL query specifies the way of building a new data source on the basis of defined in the snapshot form tabular source data.

2.2.1.1.2 Inner data source – source reports type.

The new inner data source allows the selection of the type of source reports.



New Inner data source

Basic data

Code: FINREP

Name: FINREP

Taxonomy set: TS[F.72.1]

Source reports type: All reports

Relative period

...
All reports
All reports from current period
Report from current period - current bank entity
Report from relative period - current bank entity
Save
Cancel

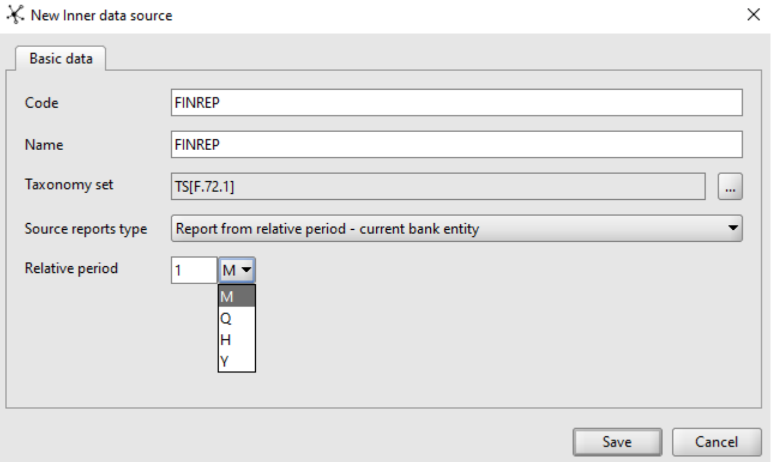
Figure 51. Source reports types

Options available in this field:

- **All reports** - all reports for the selected taxonomy set will be available as a source for data fetching.
- **All reports from current period** - all reports will be available within the selected set of taxonomies, but only for the period matching the period for which the report is created.
- **Report from current period - current bank entity** - one report of the current period report will be available, for the selected set of taxonomies and the banking unit matching.
- **Report from relative period - current bank entity** - a single report will be available for the selected set of taxonomies and banking unit matching with the report, with reference to the relative period set in the data source definitions.

Relative period can be set for:

- M – months,
- Q – quarters,
- H - half-years,
- Y – years.



Code: FINREP
Name: FINREP
Taxonomy set: TS[F.72.1]
Source reports type: Report from relative period - current bank entity
Relative period: 1 M

Figure 52. Relative periods

aSISt will automatically search for the indicated reports for the unit for which the report was created and for the period calculated as the date of the report **minus the value entered in the field**, in relation to the selected period type (months/quarters/half-years/years).

*Example: by setting the values **1** and **M** in the 'Relative period' field, we will be able to fetch the August monthly report with data from the July report. For values **4** and **M**, the August report will be fetched with data from the April report.*

2.2.1.1.3 Parameterized data source

With the use of the Parameters bookmark:

- it is possible to defined parameters used in parameterizing a data source,
- the screen displays information as on Figure 51.

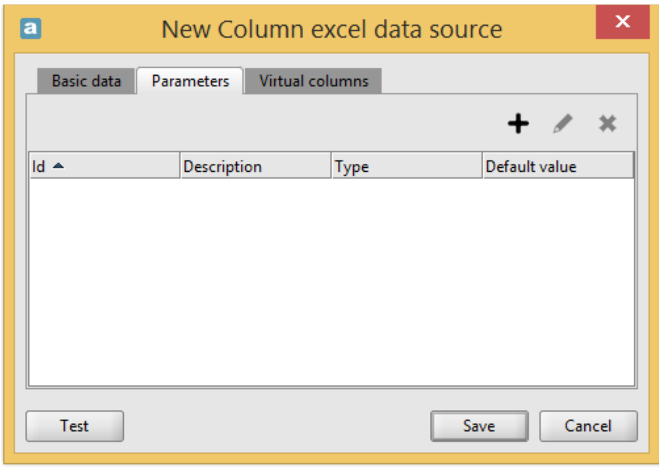
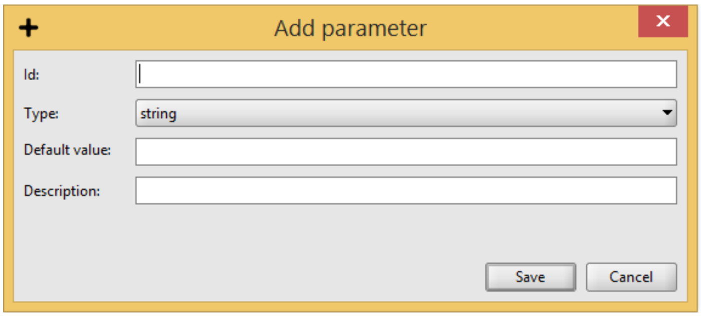


Figure 53. Managing parameters in parameterizing a data source

Selection of + bookmark allows for:

- creating a new parameter used in parameterizing a data source,
- the screen displays information as on Figure 52.



Id:
Type: string
Default value:
Description:

Figure 54. Defining a new parameter for the parameterized data source

It is possible here to define parameters of the following types:

- textual,
- numerical,
- logical,
- date.

When managing parameters for parameterized data source, the following additional functional buttons are used:

✎ – allowing for editing a previously defined parameter,

✖– allowing for removal of a previously defined parameter,

If the parameters defined here are used in specifying the access path to the data source, then – while data fetching – the system each time refers to the parameters defined here.

2.2.1.1.4 *Creating virtual columns for data sources*

With the use of the Virtual columns bookmark:

- all tabular data sources allow for creation of "virtual columns" (which do not exist in our data source) for which mapping expressions are defined,
- the screen displays information as on Figure 53.

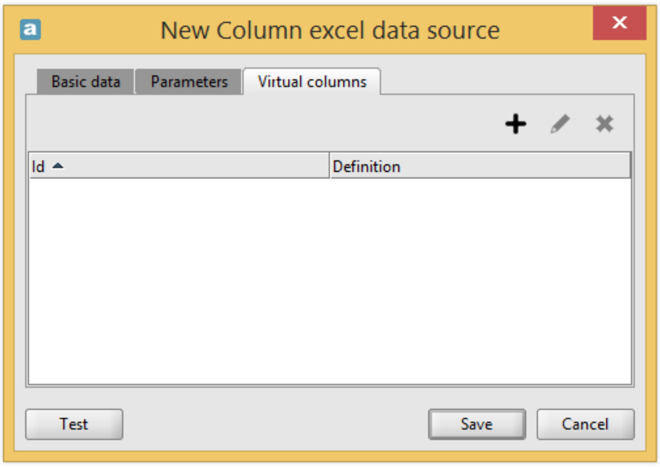


Figure 55. Managing virtual columns for data sources

Selection of + bookmark allows for:

- creation of a virtual column,
- defining a mapping expression for it,
- the screen to display information as on Figure 54.

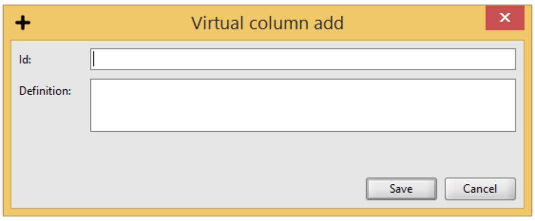


Figure 56. Adding virtual columns to data sources

Virtual columns need to have unique identifiers within the source.
Virtual column identifiers overlapping with the column name in the source table cause an error when data source is used.

Virtual column definition is an expression which allows for enumeration of values for each record. Definition can use columns in source table but cannot use other virtual columns.

A virtual column expression can use all constructions of the data source language except for the aggregate functions – it means that the virtual column value for a specified record (row) can depend solely on the value of other fields in this record.

$balance * $exchangeRate

Column identifier allows for reference to its value in mapping expressions in the same way as in the remaining data source columns:

sum($virtual_column)

Virtual columns created here can simplify mapping expressions up to a great extent, allowing to obtain indirect values for each record of source data.

In virtual columns management, additional functional buttons are used:

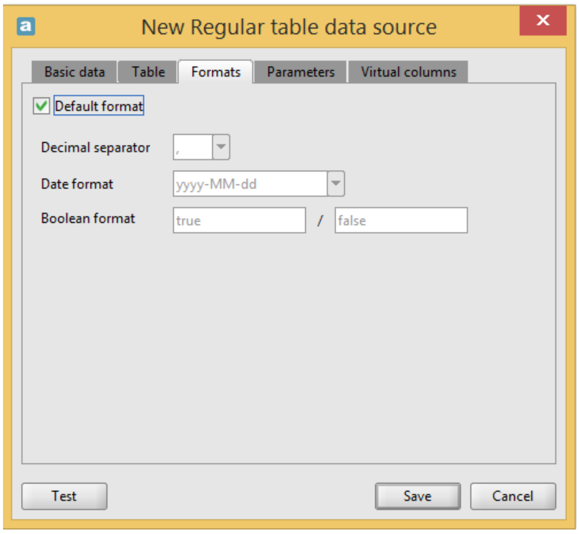
✎ – allowing for editing a previously defined virtual column,

✖ – allowing for removal of a previously defined virtual column.

2.2.1.1.5 CSV and Regular data source formats

With the use of the Formats bookmark:

- the user can implement changes in the CSV, Regular and tabular data source with dictionaries types of data source format;
- the screen displays information as on Figure 55.



☑ Default format

Decimal separator ,

Date format yyyy-MM-dd

Boolean format true / false

Figure 57. Formats for Regular data source type

After the ☑ Default format is switched off, i.e. it has the following form □ Default format, there is a possibility of implementing changes within formats used in the data source:

- decimal separators ('.' or ','),
- date formats:
  - yyyy-MM-dd,
  - yyyy.MM.dd,
  - dd-MM-yyy,
  - dd.MM.yyy,
  - other provided by the user,
- logical formats:

'<a>|<b>' where a and b are true and false value respectively.

2.2.1.2 Edit data source

To execute **"Edit data source",** in the **"Data source management"** bookmark select:

- **"Data source" menu** ✎ Edit data source

or

- icon: ✎.

This function:

- allows for implementing changes to a previously defined data source,
- refers always to the marked (highlighted) data source on the list,

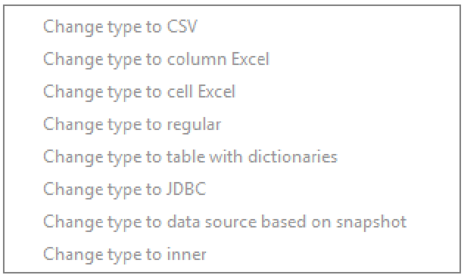
- displays the screen as in defining a new data source, however, in this case it includes also previously saved information which can be modified,
- does not allow for implementation of changes in the data source identifier.

2.2.1.3 Change data source type

To execute **"Change data source type",** in the **"Data source management"** bookmark select:

- **"Data source" menu** Change type

and then one of the appropriate types:


Change type to CSV
Change type to column Excel
Change type to cell Excel
Change type to regular
Change type to table with dictionaries
Change type to JDBC
Change type to data source based on snapshot
Change type to inner

This function:

- allows for implementing changes of the data source type to a correct one in the earlier defined data source (highlighted),
- does not cause conversion of the specified file to a new type,
- can be used if in defining the data source an incorrect type was selected by mistake.

2.2.1.4 Data source history

To execute **"Data source history",** in the **"Data source management"** bookmark select:

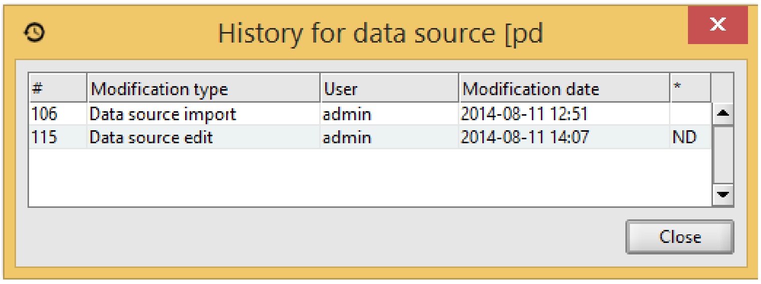
- **"Data source" menu** 🕓 Data source history

or

- icon 🕓.

This function:

- displays information about changes introduced to the selected (highlighted) data source,
- displays the history of implemented changes on the screen, as on Figure 56.



#	Modification type	User	Modification date	*
106	Data source import	admin	2014-08-11 12:51	
115	Data source edit	admin	2014-08-11 14:07	ND

Figure 58. Change history of data source definitions

2.2.1.5 Import data sources

To execute **"Import data source definitions",** in the **"Data source management"** bookmark select:

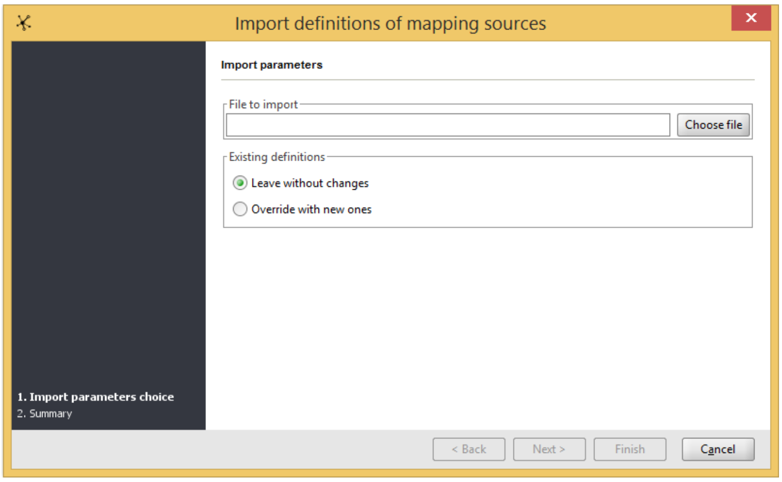
- **"Data source" menu** Import data sources

or

- icon .

This function:

- allows for importing data source definitions,
- displays information as on Figure 57.



Import parameters

File to import

Existing definitions

● Leave without changes

○ Override with new ones

1. Import parameters choice

2. Summary

Figure 59. Import mapping source definitions

2.2.1.6 Export data sources

To execute **"Export data sources"**, in the **"Data source management"** bookmark select:

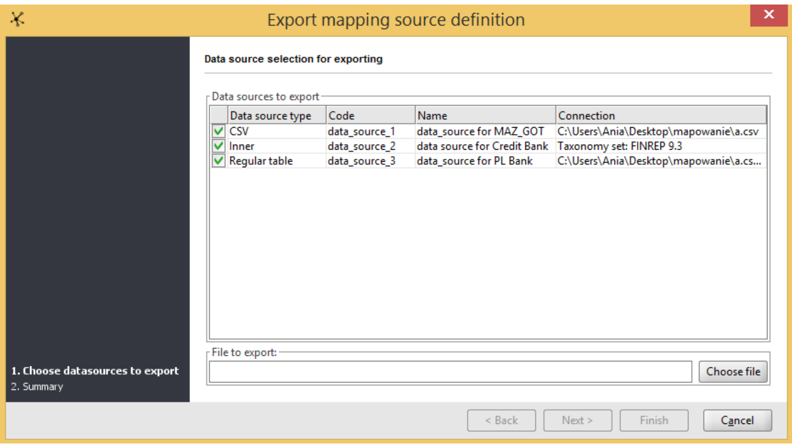
- **"Data source" menu** Export data sources

or

- icon ⤒.

This function:

- allows for exporting data source definitions,
- displays information as on Figure 58.



Export mapping source definition

Data source selection for exporting

Data sources to export

	Data source type	Code	Name	Connection
☑	CSV	data_source_1	data_source for MAZ_GOT	C:\Users\Ania\Desktop\mapowanie\a.csv
☑	Inner	data_source_2	data source for Credit Bank	Taxonomy set: FINREP 9.3
☑	Regular table	data_source_3	data_source for PL Bank	C:\Users\Ania\Desktop\mapowanie\a.cs...

File to export:

Figure 60. Export mapping source definitions

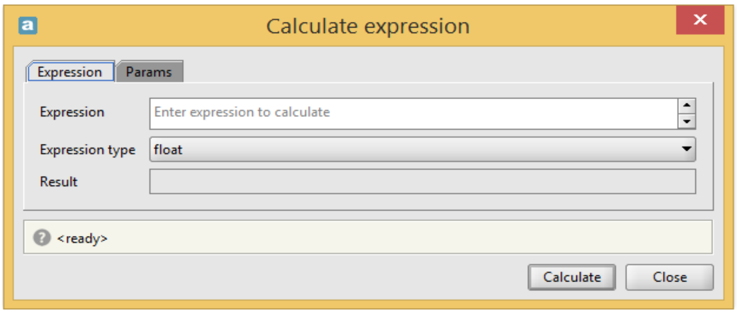
2.2.1.7 Calculated expressions

To execute **"Calculate expression",** in the **"Data source management"** bookmark select:

- **"Data source" menu** Calculate expression

This function:

- allows for verification of correctness of defined mapping expressions and the data fetching sources,
- quick calculation of data on the basis of formulated expression and data source,
- displays information as on Figure 59.



Calculate expression

Expression: Enter expression to calculate

Expression type: float

Result:

<ready>

Figure 61. Quick verification of defined expressions for data fetching

After

- entering in the row:
  - an expression for data fetching,
- selecting an appropriate expression type:
  - numerical,
  - logical,
  - text,
- accepting entered information with the Calculate button,

on the basis of the provided data source in the expression, **system will verify and calculate data**.

In case there occur errors in calculation, the user will be informed about this with an appropriate message.

3 "Snapshot warehouse" module

To launch **"Snapshot module"** select:

⚙ Snapshot storage

or

+ – located on the module bookmarks tab (see Figure 60), and from the displayed list select: Snapshot storage .

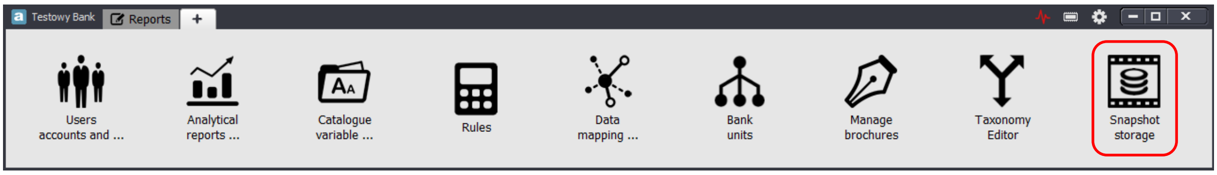


Figure 62. aSISt module bookmarks bar – with the highlighted "Snapshot warehouse" module

A new module bookmark will be created Snapshot storage :

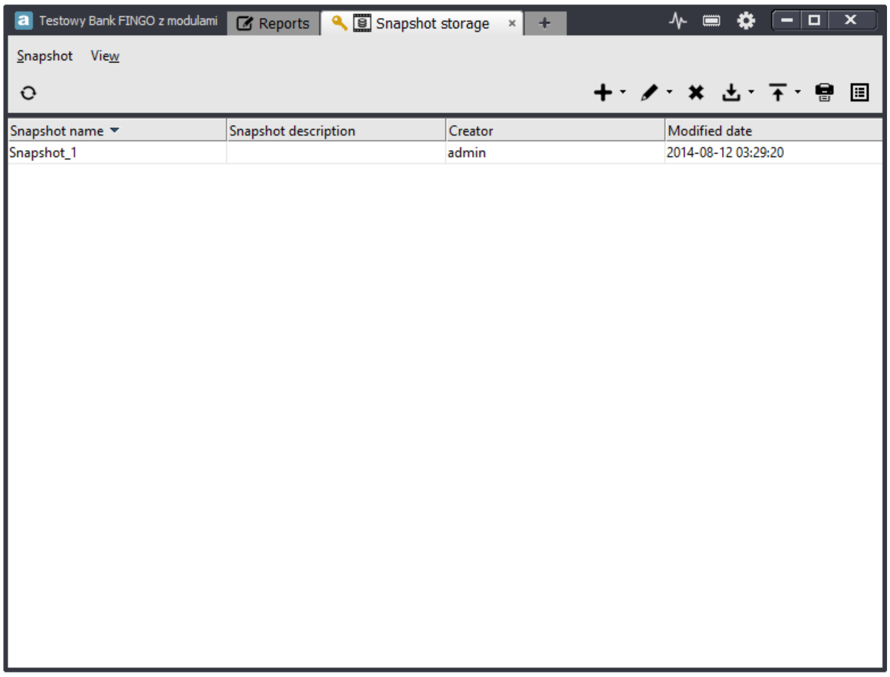
- which is closely linked with the mappings module,

- owing to which a significant extension of mapping module possibilities is obtained – the module is extended by snapshot data source,
- which facilitates:
  - creating and managing snapshots which are sets of tables built on the basis of source data and SQL transformations stored in an independent database built into aSISt,
  - storing source data individually for each reporting period in the form of snapshot instances.

After "Snapshot warehouse" is activated (see Figure 61):

- a list of all previously defined snapshot definitions is displayed,
- the following functions are available:
  - grouping in the "Snapshot" and "View" menu.

The functions are displayed in the form of functional icons.



Snapshot name	Snapshot description	Creator	Modified date
Snapshot_1		admin	2014-08-12 03:29:20

Figure 63. Snapshot warehouse – snapshot list display

"Snapshot warehouse" module:

- is available only for banks with an extended license (including this module),
- in multiple-user version can be simultaneously used only on one workstation.

Snapshots:

- are definitions of tabular source data, used in the mappings module as data source snapshots,

- can consist of multiple various, independent tables which are subject to data fetching from back office systems;
- are not linked with any reporting periods,
- do not define any links between different source tables,
- are identified by their unique names.

Displayed list of snapshot definitions:

- provides information about:
  - snapshot name,
  - description applied,
  - author login,
  - modification date,
- for a greater visibility can be sorted by any column.

More information about defined snapshots can be obtained by sorting them while using the "Edition of snapshot definitions" function (see chapter 3.1.3 Edit snapshot definitions).

All previously defined snapshots are displayed in the snapshot warehouse in the same way, irrespective of the method of defining them, i.e.:

- "Add on the basis of source data" (see chapter 3.1.1 Add on the basis of source data),
- "Add without source data" (see chapter 3.1.2 Add without source data).

3.1 Snapshot

"Snapshot" menu groups functions (see Figure 62) **allowing for manipulations with snapshot definitions**, i.e.:

- creating new snapshots,
- editing definitions of snapshots previously created,
- editing ETL transformations,
- deleting unnecessary and incorrectly defined snapshots,
- snapshots import/export,
- managing snapshots instances.

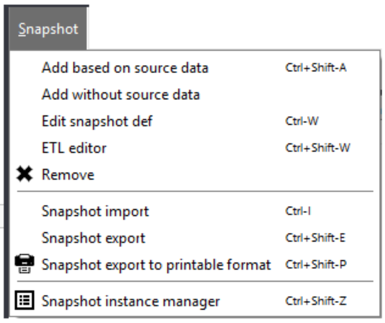

Snapshot
Add based on source data Ctrl+Shift-A
Add without source data
Edit snapshot def Ctrl-W
ETL editor Ctrl+Shift-W
Remove
Snapshot import Ctrl-I
Snapshot export Ctrl+Shift-E
Snapshot export to printable format Ctrl+Shift-P
Snapshot instance manager Ctrl+Shift-Z

Figure 64. "Snapshot" menu in "Snapshot warehouse" module

Except for the import function and creating new snapshots, all remaining functions are activated only after highlighting (marking) on a list a snapshot for which the user wished to activate the required function.

3.1.1 Add on the basis of source data

To create a **"New snapshot on the basis of source data"** select:

- **"Snapshot" menu** Add based on source data

or

- icon +,

and then option Add based on source data

or

- keyboard shortcut **Ctrl+Shift-A**.

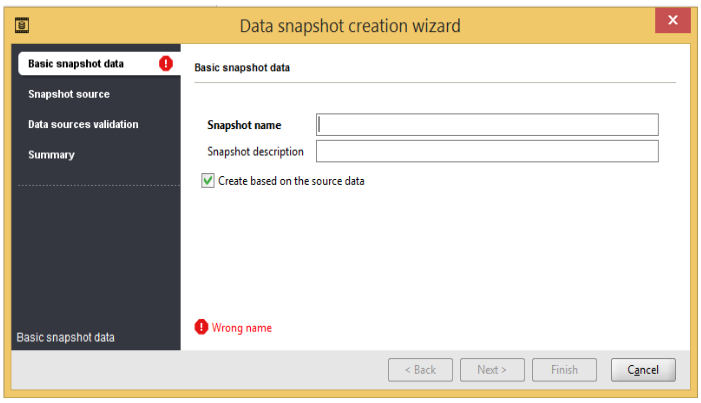
This function:

- on the basis of tabular source data saved in the CSV format allows for quick snapshot defining.

Snapshot creator each time consists of 4 steps.

Step 1 – specifying basic data of the created snapshot, i.e. data fetching (see Figure 63):

- snapshot name (max 125 characters),
- snapshot description (max 255 characters) – optional.



Data snapshot creation wizard

- Basic snapshot data
- Snapshot source
- Data sources validation
- Summary

Basic snapshot data

Basic snapshot data

Snapshot name

Snapshot description

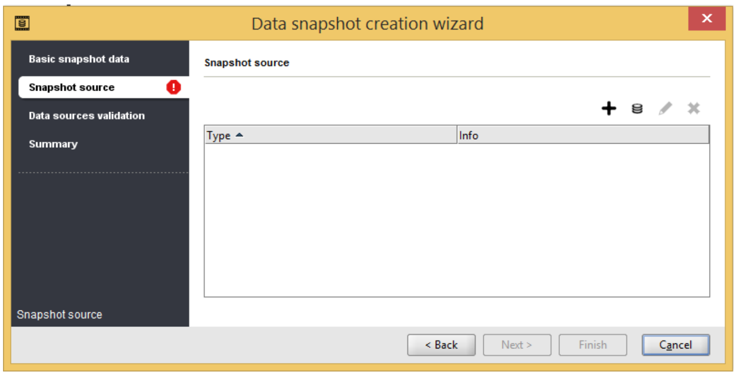
☑ Create based on the source data

Wrong name

< Back | Next > | Finish | Cancel

Figure 65. Selection of basic data while creating a snapshot

Step 2 – specifying sources for the created snapshot (see Figure 64):



Data snapshot creation wizard

- Basic snapshot data
- Snapshot source
- Data sources validation
- Summary

Snapshot source

Snapshot source

Type ▲	Info

< Back | Next > | Finish | Cancel

Figure 66. Specifying sources for the newly created snapshot

With the available functional buttons it is possible:

✚ – to specify:

- ○ file,
- ○ catalog,

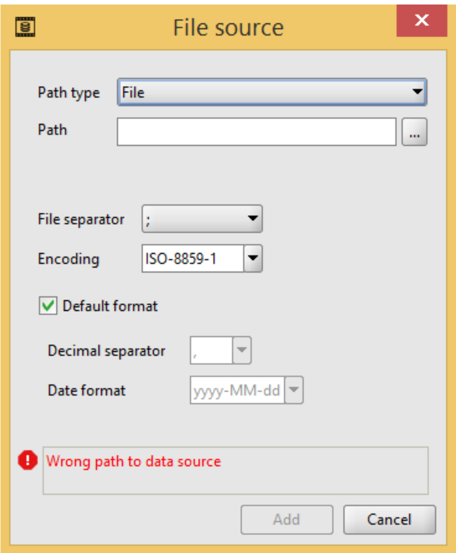
🛢 – to specify tables from database,

✎ – to edit previously specified source,

✖ – to remove previously specified source.

After selecting **"Add file source"** by clicking the functional button ✚ the screen displays information as on Figure 65:

- after selecting data source path type:
  - ○ file,
  - ○ catalog,
- it is necessary to use the buttor ... , and specify an access path of the source.



Path type: File

Path:

File separator: ;

Encoding: ISO-8859-1

☑ Default format

Decimal separator: ,

Date format: yyyy-MM-dd

Wrong path to data source

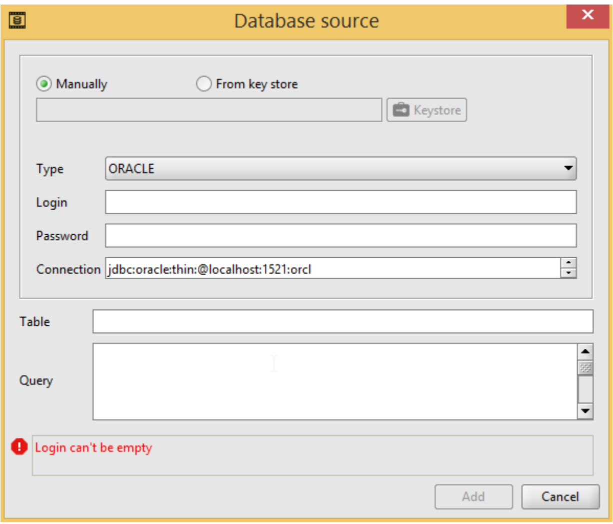
Figure 67. Defining file source

Additionally, it is possible to implement changes in the displayed settings:

- file separator (; , I <TAB>)
- coding type:
  - ISO 8859-1
  - ISO-8859-2
  - Windows-1250
  - UTF-8
  - UTF-16
  - US-ASCII
- decimal separator (, or .)
- date format:
  - yyyy-MM-dd
  - yyyy.MM.dd
  - dd-MM-yyyy
  - dd.MM.yyy

After selecting **"Add database source"** by clicking the functional button the screen displays information as on Figure 66; it is necessary here to enter the following information:

- table name,
- address (JDBC URL),
- database access data (login, password),
- SQL query.



● Manually ○ From key store

Type: ORACLE

Login:

Password:

Connection: jdbc:oracle:thin:@localhost:1521:orcl

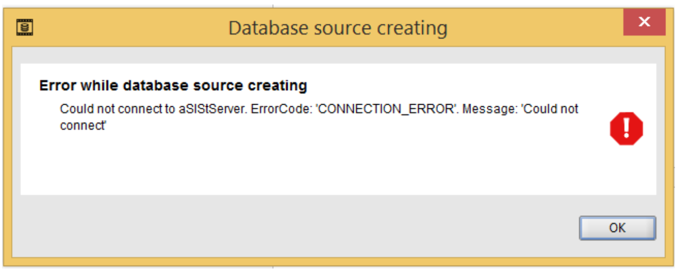
Table:

Query:

Login can't be empty

Figure 68. Adding a database source

In case incorrect data are entered, a message may appear, as on Figure 67.

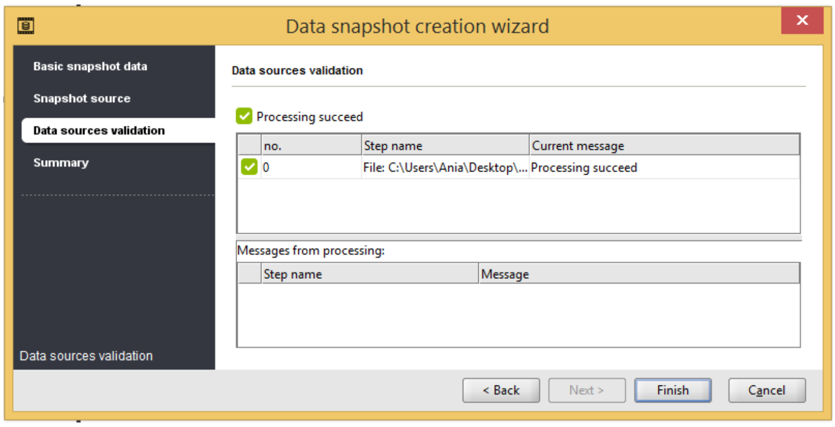


Error while database source creating

Could not connect to aSIStServer. ErrorCode: 'CONNECTION_ERROR'. Message: 'Could not connect'

Figure 69. Message about data errors in the created database source

Step 3 – initial validation of selections performed (see Figure 68).



Data sources validation

☑ Processing succeed

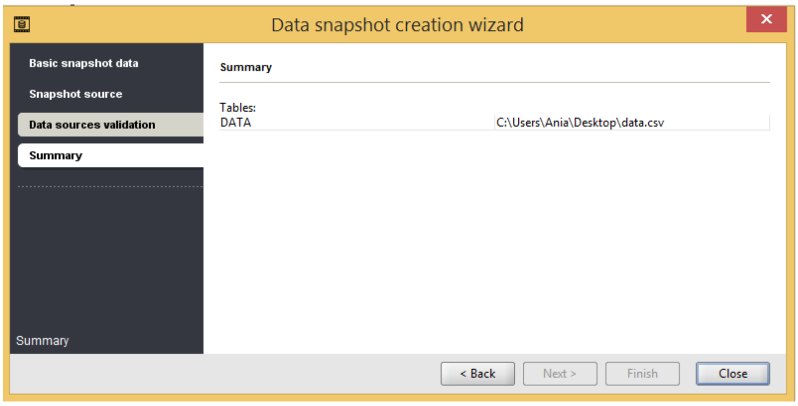
no.	Step name	Current message
☑ 0	File: C:\Users\Ania\Desktop\...	Processing succeed

Messages from processing:

Step name	Message

Figure 70. Initial validation of selected source data

Step 4 – displaying tables with which the created snapshot was linked (see Figure 69).



Summary

Tables:

DATA C:\Users\Ania\Desktop\data.csv

Figure 71. Displaying tables with which the created snapshot was joined

Selecting Close button will:

- end the function performance,
- save the defined snapshot in accordance with selections made during its creation,
- add the newly created snapshot to the list of defined snapshots (see Figure 61).

3.1.2 Add without source data

To create a **"New snapshot without modeling on source data"** select:

- **"Snapshot" menu** Add without source data

or

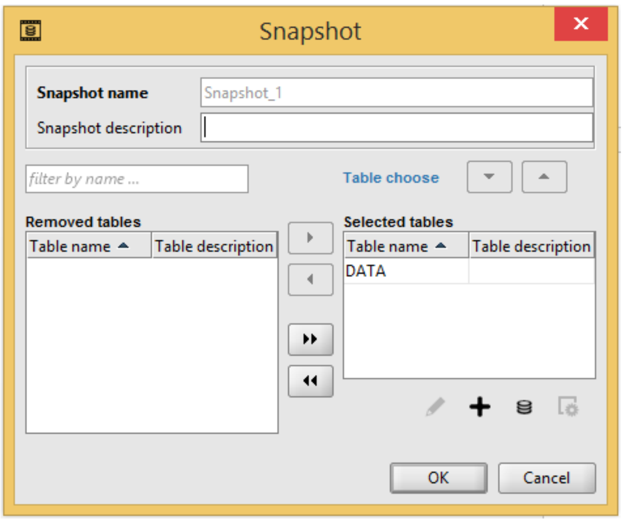
- icon: ✚

and then the option Add without source data .

This function:

- requires manual defining of tables with which the created snapshot is to be linked,
- is used when the created snapshot refers to multiple various source data and for this reason defining snapshot on the basis of one source is not convenient.

When this function is launched, the screen displays information as on Figure 70.



Snapshot

Snapshot name	Snapshot_1
Snapshot description	

filter by name ...

Table choose

Removed tables

Table name	Table description

Selected tables

Table name	Table description
DATA	

OK Cancel

Figure 72. Creating a new snapshot without indicating source data

Apart from basic data, such as:

- snapshot name (max 125 characters),
- snapshot description (max 255 characters) – optional,

in this process it is **necessary** to:

- define tables linked with the created snapshot (see chapter 3.1.2.1 3.1.2.1 Add new table),

it is **possible** to:

- enter changes to incorrectly defined tables (see chapter 3.1.2.3 3.1.2.3 Edit table),
- define and manage indexes which facilitate faster execution of SQL queries (see. chapter 3.1.2.4 3.1.2.4 Manage indexes).

A list of all defined tables is always displayed in the **right window panel**.
All tables which are linked with the created snapshot should be included in the list.

After the table is defined, the button OK is activated and it is possible to save the created snapshot and display it on the list, as shown on Figure 61.

3.1.2.1 Add new table

To define the snapshot using **"Add without source data"** function it is necessary to define all tables linked with this snapshot using **"Add table"** function (see Figure 70), i.e. selecting:

- **"Snapshot" menu** Add without source data

or

- icon +

and option Add without source data

and then

- functional button + .

This function:

- allows for defining a new table,
- displays information as on Figure 71.

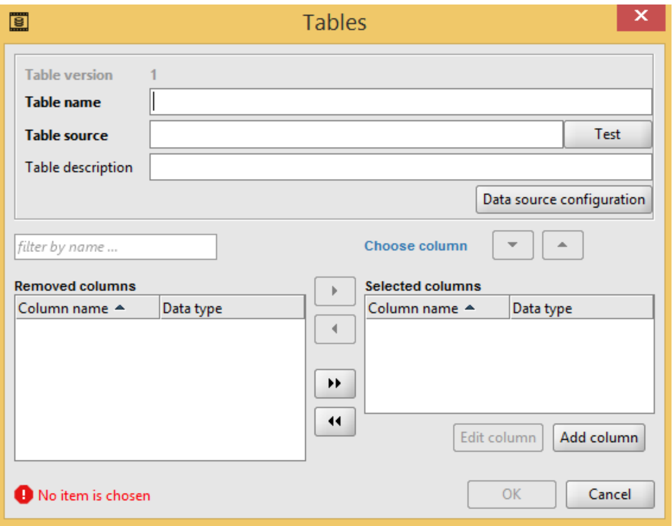


Table version 1

No item is chosen

Figure 73. Defining the table for a newly created snapshot manually

It is necessary to provide in this case:

- table name (max 125 characters),
- table source,
- table description (max 255 characters) – optional,
- detailed information about columns in the table (see Figure 71).

For the specified data source it is necessary to also provide configuration data of this tabular data source.

3.1.2.1.1 Source data configuration

To set an **appropriate data source configuration** the user needs to perform **"Source data configuration"** by selecting:

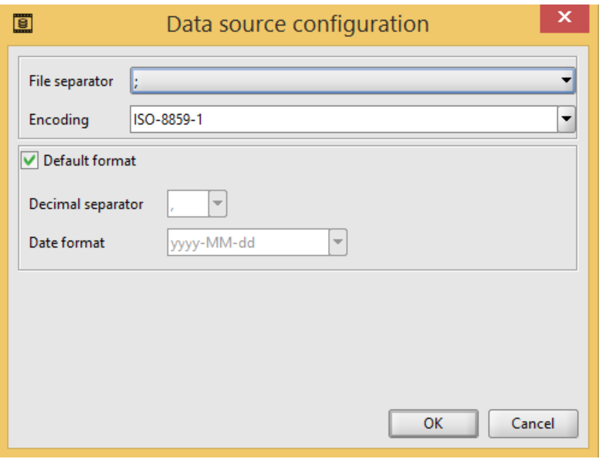
- the functional buttor Data source configuration .

Then a screen as on Figure 72 will appear on which the user can enter changes with respect to:

- file separators used:

  - ;
  - ,
  - |
  - <TAB>
- character coding type:
  - ISO 8859-1
  - ISO-8859-2
  - Windows-1250
  - UTF-8
  - UTF-16
  - US-ASCII
- decimal separator used:
  - , (default)
  - .
- date format:
  - yyyy-MM-dd (default)
  - *yyyy-MM.dd*
  - *dd-MM-yyy*
  - *dd.MM.yyy*

Changes in the decimal separator and date format can be implemented after the following option is disabled: ☑ Default format , i.e. the option has the following for □ Default format .


Data source configuration

File separator ;

Encoding ISO-8859-1

☑ Default format

Decimal separator ,

Date format yyyy-MM-dd

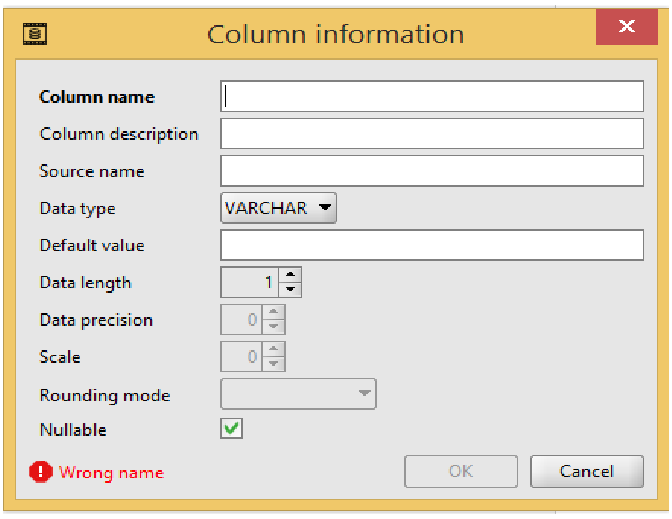
Figure 74. Data source configuration for the defined table

3.1.2.1.2 *Add column*

To **define columns** in the tabular data source linked with the created snapshot the user needs to execute **"Add column"** by selecting:

- functional button Add column .

Then the screen will display information as on Figure 73.



Column information

Column name

Column description

Source name

Data type VARCHAR

Default value

Data length 1

Data precision 0

Scale 0

Rounding mode

Nullable ☑

Wrong name

Figure 75. Information about the column of the defined table

This function requires:

- table name (max 125 characters),
- name in the source,

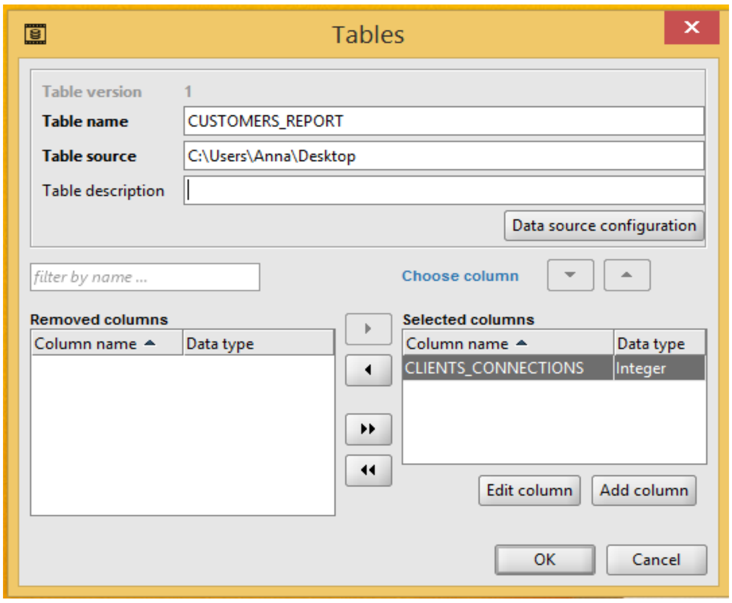
and depending on the data type selected, selection of an appropriate:

- data length,
- data precision.

Data type can the selected as:

- VARCHAR – character-based with specified length,
- INTEGER – average integer,
- SHORT – short integer,
- LONG – long integer,
- DECIMAL – fixed-point,
- DATE – date type.

Confirming entered settings with the button OK will save the defined column in the **right panel** of the displayed window, as on Figure 74.



Tables

Table version 1

Table name: CUSTOMERS_REPORT

Table source: C:\Users\Anna\Desktop

Table description:

Selected columns:

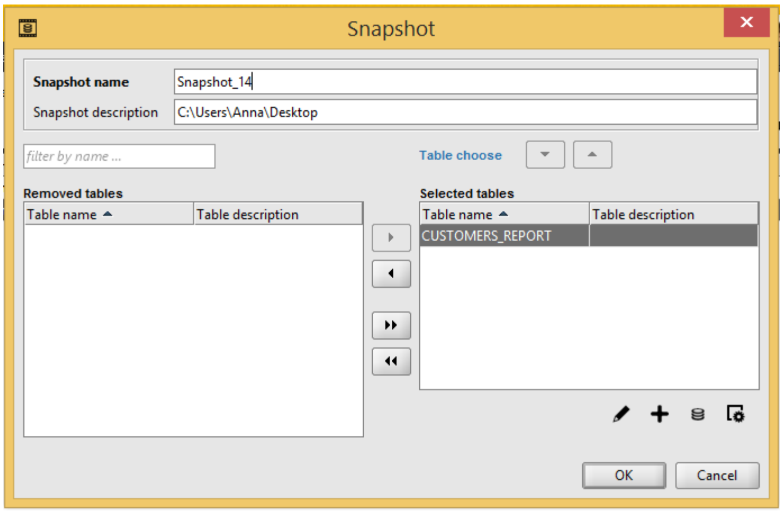
Column name	Data type
CLIENTS_CONNECTIONS	Integer

Figure 76. List of columns selected for the defined table

It is necessary in this case to define all columns for the defined table of the created snapshot and confirm selections made with the button OK .

Application:

- will return then to the screen on which a list of tables linked with the created snapshot is displayed, as on Figure 75 (compare with Figure 70),
- will enable adding another table to the snapshot created.



Snapshot

Snapshot name: Snapshot_14

Snapshot description: C:\Users\Anna\Desktop

Selected tables:

Table name	Table description
CUSTOMERS_REPORT	

Figure 77. Defining snapshots – defining and saving the required snapshot

3.1.2.1.3 Remove column

If for any reason, on the list of defined columns, there are columns which should not be linked with the given table, it is possible to **remove** them.

To remove previously defined columns the user needs to:

- move the columns to be removed from the right to the **left window panel**.

It can be performed with the use of the functional buttons:

◂ – removal of one or several marked table columns,

◂◂ – removal of all defined columns.

Removal of defined columns and their move to the left window panel does not need to be a permanent action – at any time it is possible to include them again, using the following buttons:

▸▸ – include all columns moved previously to the left window panel,

▸ – include selected (marked) columns moved previously to the left window panel.

3.1.2.1.4 Edit column

Defined columns, located on the list in the right window panel (see Figure 74) **can be modified** with the use of the **"Edit columns"** function by using:

- the functional button Edit column .

This function:

- is activated after the column for which changes are to be implemented is highlighted (marked),
- displays data entered previously,
- allows for implementation of changes in all displayed positions.

3.1.2.2 Remove table

If for any reason on the list with tables selected to the created snapshot there are tables which should not be linked with the created snapshot, it is **possible to remove them** (in accordance with the same rule applied to removal of incorrect columns).

To remove previously defined tables the user needs to:

- move the tables to be removed from the right to the **left window panel**.

Functional buttons used here execute the following actions:

◀ – removal of one or several marked tables,

◀◀ – removal of all defined tables.

Exclusion of tables from the snapshot and their move to the left window panel does not need to be a permanent action – at any time it is possible to include them again, using the following buttons:

▶▶ – include all tables moved previously to the left window panel,

▶ – include selected (marked) tables moved previously to the left window panel.

3.1.2.3 Edit table

To define the snapshot using **"Add without source data"** function it is possible to make changes in the defined but not saved tables using **"Edit table"** function (see Figure 75), i.e. selecting:

- **"Snapshot" menu** Add without source data

or

- icon +

and option Add without source data

and then, after the table is defined:

- highlighting the table for which changes were made in the right window panel

and selecting

- the functional button ✎ .

This function:

- is activated after the table for which changes are to be implemented is highlighted (marked) in right window panel,
- displays screen with previously entered data (see Figure 74),
- allows for modifications of all positions.

3.1.2.4 Manage indexes

For faster executions of defined SQL queries it is possible to use the **"Manage indexes"** function and assign additional indexes to selected columns in tabular source data.

This function can be activated while defining snapshots with the **"Add without source data"** function (see Figure 67) by selecting:

- **"Snapshot" menu** Add without source data

or

- icon ✚,

and option Add without source data ,

and then, after the table is defined:

- highlighting the table for which "Manage index" function is to be activated in the right window panel

and selecting

- the functional button ⚙ .

This function:

- is activated after the table for which "Manage indexes" function is to be activated is highlighted (marked) in right window panel,
- allows for defining and removal of incorrect indexes,
- displays information as on Figure 76.

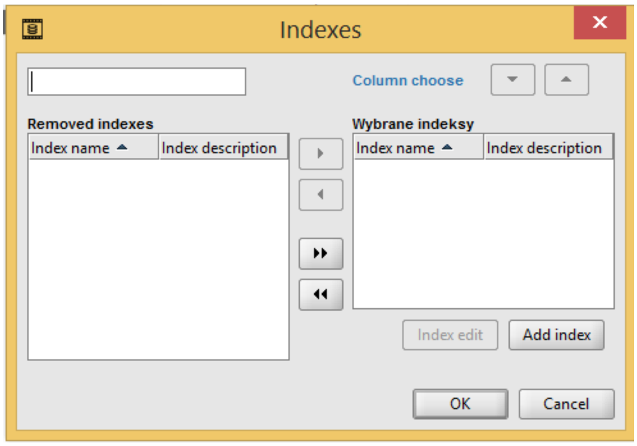


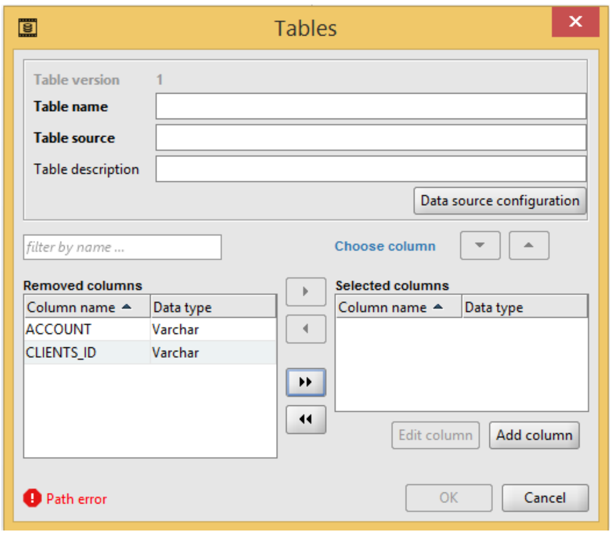
Figure 78. Managing indexes

3.1.2.1.5 *Add index*

To **define indexes for columns of the selected table** the user needs to launch **"Add index"** function, selecting:

- functional button Add index .

Then the screen will display information as on Figure 77.



Tables

Table version 1
Table name
Table source
Table description

Removed columns

Column name	Data type
ACCOUNT	Varchar
CLIENTS_ID	Varchar

Selected columns

Column name	Data type

Path error

Figure 79. Index column list

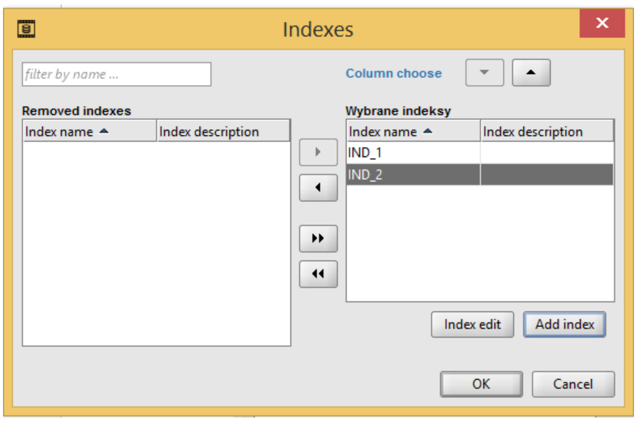
Displayed on screen in the **left window panel** are all previously defined, available columns for the selected table.

To define indexes, it is necessary in this case to:

- provide a name of the created index (max .... characters),
- select an appropriate index type:
  - unique
  - non-unique
- move from the left to the right window panel those columns for which the created index is to be assigned.

Confirming the selection made with the OK button will result in:

- assigning additional indexes to selected columns,
- saving defined indexes for the given snapshot,
- displaying created indexes in the right panel of the screen as on Figure 78.



Removed indexes

Index name	Index description

Wybrane indeksy

Index name	Index description
IND_1	
IND_2	

Figure 80. Managing indexes – list of created indexes

3.1.2.1.6 *Remove index*

To remove previously defined index the user needs to:

- move the selected index from the right to the **left window panel**.

It can be performed with the use of the functional buttons:

◀ – removal of one or several marked indexes,

◀◀ – removal of all defined indexes.

Removal of defined indexes and their move to the left window panel does not need to be a permanent action – at any time it is possible to include them again, using the following buttons:

▶▶ – include all indexes moved previously to the left window panel,

▶ – include selected (marked) indexes moved previously to the left window panel.

3.1.2.1.7 *Edit index*

Defined indexes, located on the list in the right window panel (see Figure 78), **can be modified** with the use of the **"Edit index"** function by selecting:

- the functional button Index edit .

This function:

- is activated after the index for which changes are to be implemented is highlighted (marked),
- displays data entered previously,
- allows for implementation of changes in all displayed positions.

3.1.3 Edit snapshot definitions

To make changes to the earlier defined snapshot the user needs to choose the **"Edit snapshot definitions"** function and then select:

- **"Snapshot" menu**
Edit snapshot def

or

- icon ✎

and option
Edit snapshot def

or

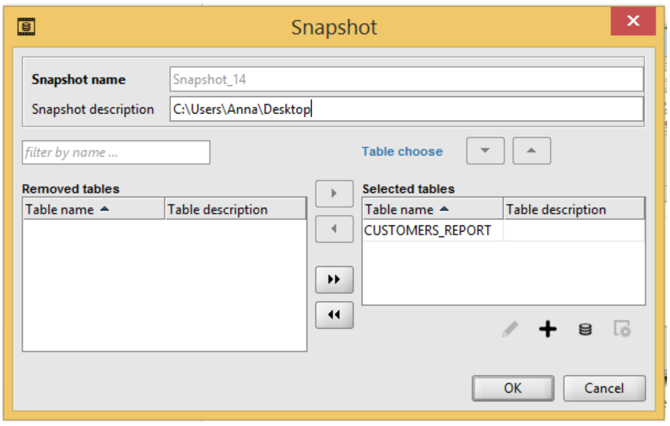
- keyboard shortcut **Ctrl-W**.

This function:

- allows for making changes in definitions of created snapshots,
- is activated when the snapshot for which modifications are to be made is marked (highlighted) on the list,
- delivers detailed information about defined and saved snapshots,
- refers to all defined snapshots, irrespective of the way they were created: on the basis of source data or without source data.

After it is launched:

- the screen displays data entered as on Figure 79 (compare Figure 75),
- it is possible to:
  - add new tables,
  - remove incorrect tables,
  - add again the tables which were previously removed (located in the left window panel),
  - make changes to the selected tables, displayed in the right panel of table window
- snapshot description can be changed,
- snapshot name can be changed.



Snapshot name: Snapshot_14

Snapshot description: C:\Users\Anna\Desktop

Table name	Table description
CUSTOMERS_REPORT	

Figure 81. Editing snapshot definition

3.1.3.1 Add new table

To extend the previously defined snapshot with a new tabular data source the user need to select **"Add table"** function by choosing:

- **"Snapshot" menu** Edit snapshot def

or

- icon ✎

and option Edit snapshot def

and then

- functional button ✚ .

This function:

- allows for defining a new table,
- when launched, displays information on screen as on Figure 71,
- works in the same way as the one launched when defining a snapshot with "Add without source data" function, described in details in the chapter 3.1.2.1 Add new table.

3.1.3.2 Remove / restore the table

To remove previously defined tables the user needs to:

- move the tables to be removed from the **right to the left window panel**,
- use one of the functional buttons:

◂ – removal of one or several marked tables,

◀◀ – removal of all defined tables.

To **restore previously defined and later removed tables**, the user needs to:

- move the tables to be restored from the **left to the right window panel**,
- use one of the functional buttons:

▶▶ – include all tables moved previously to the left window panel,

▶ – include selected (marked) tables moved previously to the left window panel.

These functions:

- works as the ones launched when defining a snapshot by the function **"Add without source data"**, described in details in the chapter 3.1.2.2 Remove table.

3.1.3.3 Edit table

To make changes to tables of the earlier defined snapshot the user needs to choose the **"Edit table"** function and then select:

- **"Snapshot" menu** Edit snapshot def

or

- icon: ✎,

and option: Edit snapshot def

and then

- highlight the table for which changes were made in the right window panel

and select

- the functional button ✎ .

This function:

- allows for modifications of all setting of previously defined tables,
- when launched, displays screen with previously entered data (see Figure 74),
- works as the one launched when defining a snapshot by the function **"Add without source data"**, described in details in the chapter 3.1.2.3 Edit table.

3.1.3.4 Manage indexes

For faster execution of SQL queries the user can add an additional index to selected columns in tabular source data using **"Manage indexes"**, by selecting:

- **"Snapshot" menu** Edit snapshot def

or

- icon ✎,

and option Edit snapshot def

and then

- highlighting the table for which "Manage index" function is to be activated in the right window panel

and selecting

- the functional button ⚙ .

This function:

- allows for defining and removal of incorrect indexes,
- when launched, displays information on screen as on Figure 78,
- works as the action launched when defining a snapshot by the function **"Add without source data"**, described in details in the chapter 3.1.3.4 Manage indexes.

3.1.4 Edit ETL

To execute "**Edit ETL"**, select:

- **"Snapshot" menu** ETL editor

or

- icon ✎

and option ETL editor

or

- keyboard shortcut **Ctrl+Shift-W**.

This function:

- allows to execute a string of transformations in the created snapshots,
- is activated when a snapshot for which modifications are to be made is marked (highlighted) on the list.

After it is launched:

- the screen displays snapshot tables defined in the selected snapshot, as on Figure 95.

Because ETL is a tool using multiple complex functions and requires an extensive description, these issues are presented in details in the additional chapter 4 ETL of this document.

3.1.5 Remove snapshot

To execute "**Remove ETL**", select:

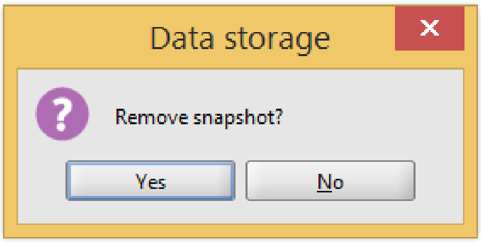
- **"Snapshot" menu** ✖ Remove

or

- icon: ✖.

This function:

- allows for **permanent removal of the marked** (highlighted) **snapshot**,
- is activated only after the required snapshot is marked (highlighted) snapshot,
- displays information as on Figure 80.



Remove snapshot?

Figure 82. Query regarding snapshot deletion

After approval:

- snapshot disappears from the snapshot warehouse, and
- it is not visible on the displayed list of existing snapshots.

3.1.6 Import snapshot

To execute "**Import snapshot**", select:

- **"Snapshot" menu** Snapshot import

or

- icon ⤓

or

- keyboard shortcut **Ctrl-I**.

This function:

- allows for import of the snapshot saved in xml format.

Full snapshot import process is performed in **three steps**.

Step 1 – specifying a path for the imported snapshot (see Figure 81):

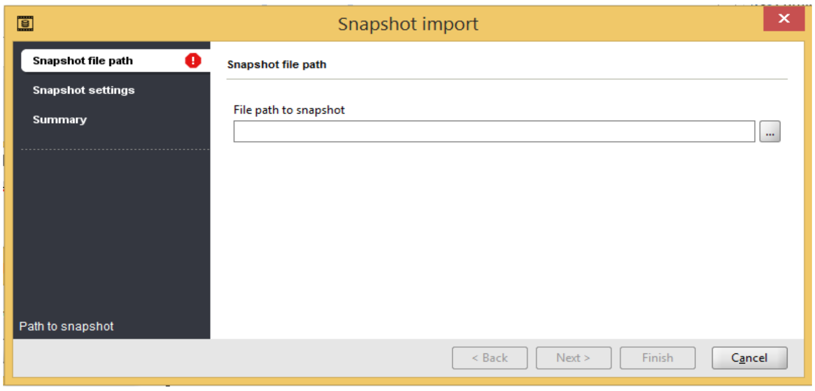
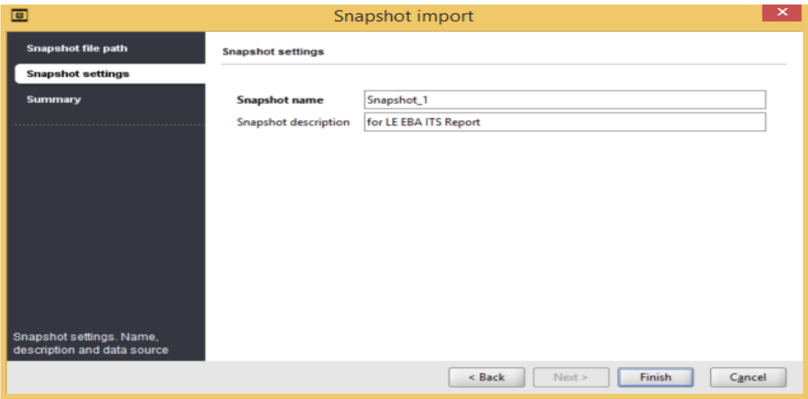


Figure 83. Specifying a path for the imported snapshot

Step 2 – displaying settings of the imported snapshot (see Figure 82):



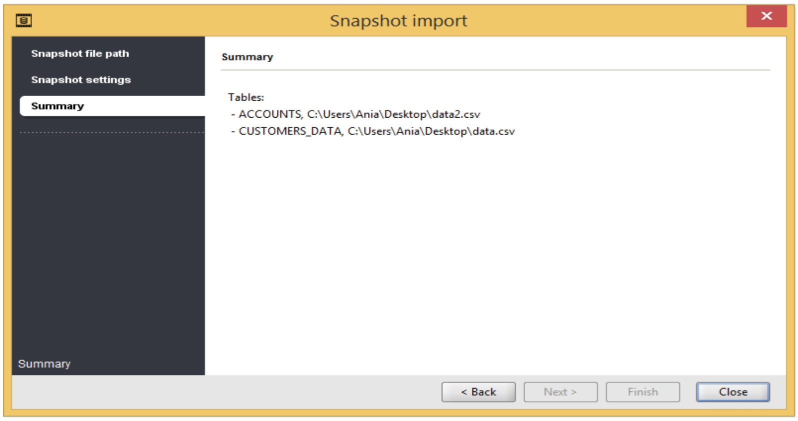
Snapshot name: Snapshot_1

Snapshot description: for LE EBA ITS Report

Snapshot settings. Name, description and data source

Figure 84. Displaying settings of the imported snapshot

Step 3 – information about tabular data source, linked with the imported snapshot (see Figure 83).



Summary

Tables:
- ACCOUNTS, C:\Users\Ania\Desktop\data2.csv
- CUSTOMERS_DATA, C:\Users\Ania\Desktop\data.csv

Figure 85. Information about tables linked with the imported snapshot

3.1.7 Export snapshot

To execute "**Export snapshot**", select:

- **"Snapshot" menu** Snapshot export

or

- icon ⤒

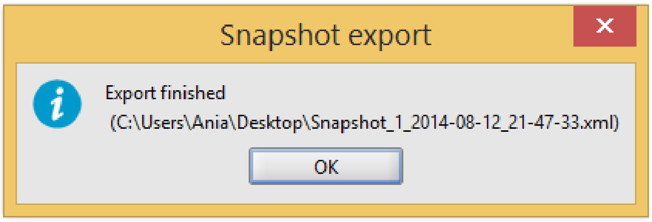
or

- keyboard shortcut **Ctrl+Shift-E**.

This function:

- allows for exporting in xml format any, previously defined snapshot,
- is activated only when on the snapshot list a snapshot to be exported is marked (highlighted),
- after it is launched, requires the user to specify the location of saving the exported snapshot.

Correctly executed export is signalled with a message as on Figure 84.



Export finished
(C:\Users\Ania\Desktop\Snapshot_1_2014-08-12_21-47-33.xml)

Figure 86. Message about the end of snapshot export

3.1.8 Manage snapshot instances

To execute "**Manage snapshot instances"**, select:

- **"Snapshot" menu** ▤ Snapshot instance manager

or

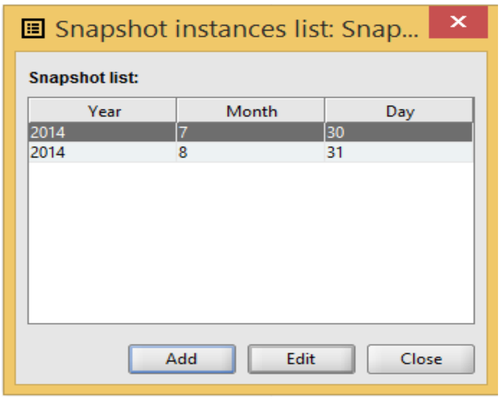
- icon ▤

or

- keyboard shortcut **Ctrl+Shift-Z**.

Functions grouped here allow for:

- displaying snapshot instances list linked with earlier selected snapshots, as on Figure 85,
- editing of instances displayed on the list,
- defining new snapshot instances.



Snapshot instances list: Snap...

Snapshot list:

Year	Month	Day
2014	7	30
2014	8	31

Figure 87. Snapshot instances list

These functions:

- can be activated only when the snapshot for which the user wants to manage its instances was previously marked on the list .

Snapshot instances:

- **are created automatically** and are linked with specific reporting periods during which, with the use of snapshot data sources, the following functions are performed:
  - "Data fetching" (described in chapter 5 Fetching by mappings),
  - "Data source test" (described in chapter 2.2.1.1.1 Basic data of the new data source),
- **can be defined manually** with the following function:
  - "Add new snapshot instance" (see chapter 3.1.8.1 3.1.8.1 Add new snapshot instance), but then they are not linked with any data,

- **can be subject to data fetching** with the following function:
  - "Data fetching" (described in chapter 5 Fetching by mappings),
  - "Data source test" (described in chapter 2.2.1.1.1 Basic data of the new data source),
  - "Load source data to marked tables" (described in chapter 3.1.8.2.5 Load source data for the marked tables).

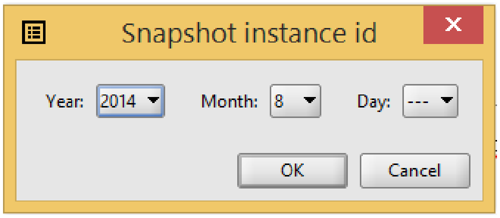
3.1.8.1 Add new snapshot instance

To define an additional snapshot instance in the selected snapshot, in **"Manage snapshot instances"** select:

- functional button Add .

This function:

- allows to create a new snapshot instance,
- when launched, displays on screen information as on Figure 86,
- requires the user to provide periods for which the new snapshot instance is to be defined.



Snapshot instance id

Year: 2014 Month: 8 Day: ---

Figure 88. Snapshot instance data

Created snapshot instance:

- is not linked with any data,

but it is possible to:

- import data to the instance,
- execute a test SQL query.

3.1.8.2 Edit snapshot instance

Snapshot instances on the list can be subject to verification with the **"Edit snapshot instances"** function; to enable the process, select:

- functional button Edit .

This function:

- displays information about the selected snapshot instance as on Figure 87,
- allows for:
  - loading selected tables linked with the snapshot with source data,
  - execution of all ETL steps,
  - initial verification of data correctness in selected tables,
  - entry and execution of SQL language queries.

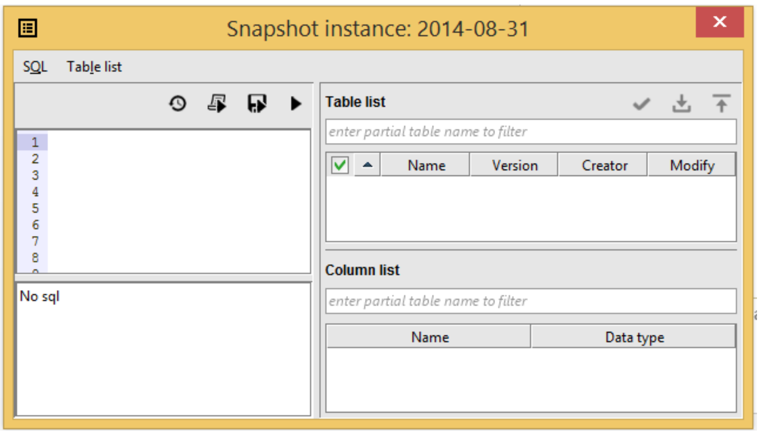

Snapshot instance: 2014-08-31
SQL
Table list
1
2
3
4
5
6
7
8
No sql
Table list
enter partial table name to filter
Name
Version
Creator
Modify
Column list
enter partial table name to filter
Name
Data type

Figure 89. Detailed information displayed in a snapshot instance

In the **left window panel** in the:

- **upper part** – it is possible to define any **SQL queries operating on data from snapshot tables and data from ETL steps** tables by referring to tables displayed in the right window panel,
- **lower part** displayed are **results of the executed SQL query**.

In the **right window panel** in the:

- upper part displayed is the **list of tables linked with the snapshot**, with the detailed information about:
  - table name,
  - snapshot definition version / ETL linked with version of data included in the instance,
  - login of table author,
  - modification date,

before the table name the square displayed is:

☐ – empty when functions performed do not refer to the given table,

☑ – marked when for the given table one of the following functions is to be performed:

- data validation or,
- source data load,

- **lower part** displays **column list for** the **table** window selected in the upper panel with the information about:

- column name,
- data type.

To mark / unmark all tables displayed in the right window panel the user needs to:

- mark / unmark the square ☑ / □ – located over the displayed table list, in the header row of the upper right window panel.

Multiple functions can be grouped in a menu:

- SQL (see Figure 88):
  - Perform,
  - Save results,
  - Execute script,
  - History,
  - Close.

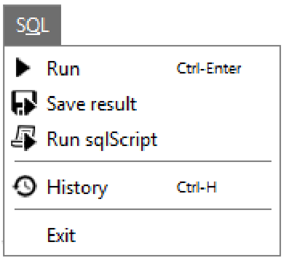


Figure 90. "SQL" menu in snapshot instances management

- Table list (see Figure 89):
  - Validate source data for the marked tables,
  - Load source data for the marked tables.

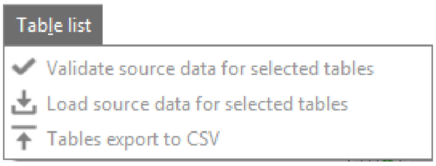


Table list
Validate source data for selected tables
Load source data for selected tables
Tables export to CSV

Figure 91. "Table list" menu in snapshot instances management

Additionally, these functions are available also as functional icons located below the functional menu:

- in the left window panel:

– history,

– execute script,

– save results,

▶ – execute,

- in the right window panel:

✓ – validate source data for marked tables,

⤓ – load source data for marked tables.

3.1.8.1.1 *SQL scripts execution*

For the purposes of testing and verification of source data linked with created snapshot instances it is possible to define and execute SQL queries referring to the table of defined snapshots.

In the "SQL" menu of the displayed snapshot instance two functions are available which allow for execution of defined queries:

- function **"Execute"** – applying to queries entered by the same user to the upper left window panel,
- function **"Execute script"** – applying to scripts provided by the aSISt supplier, in the form of zip files.

To execute a **script defined by the user**, applying to the snapshot instance tables, the user, in the editing mode of the selected instance in **"Snapshot instance management"**, needs to proceed as follows:

- to the **left upper window panel** enter an **SQL query**,

and then select:

- **"SQL" menu** ▶ Run

or

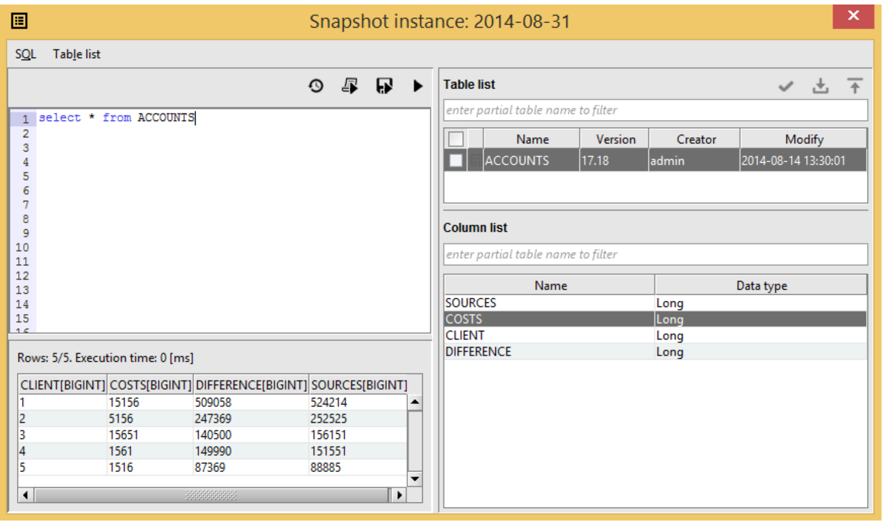
- icon ▶

or

- keyboard shortcut: **Ctrl-Enter**.

This function:

- in the left window panel displays:
  - the result of a correctly defined SQL query (as on Figure 90), or
  - information about errors in the formulated query,
- can be executed correctly only for selected tables which have the loaded source data.


Snapshot instance: 2014-08-31
SQL
Table list
Table list
enter partial table name to filter
Column list
enter partial table name to filter

```
select * from ACCOUNTS
```

	Name	Version	Creator	Modify
□	ACCOUNTS	17.18	admin	2014-08-14 13:30:01

Name	Data type
SOURCES	Long
COSTS	Long
CLIENT	Long
DIFFERENCE	Long

Rows: 5/5. Execution time: 0 [ms]

CLIENT[BIGINT]	COSTS[BIGINT]	DIFFERENCE[BIGINT]	SOURCES[BIGINT]
1	15156	509058	524214
2	5156	247369	252525
3	15651	140500	156151
4	1561	149990	151551
5	1516	87369	88885

Figure 92. Information displayed in snapshot instance after SQL query was executed

When defining an SQL query a **function allowing for simple and fast transfer of a column name or a column to the defined query** might be helpful; the function can be enabled by:

- **double-clicking with the left mouse button on the required table or column from the right window panel.**

In case help of aSISt service is required in defining complex SQL queries, the function enabling execution of provided scripts is **"Execute script"** performed in **"Snapshot instance management"** in edition of the selected instance by choosing:

- **"SQL" menu** Run sqlScript

or

- icon .

This function:

- requires specifying files with SQL scripts, saved as *.zip,
- in the lower left window panel displays the result of the defined SQL query.

3.1.8.1.2 Saving SQL query results

To save the result of the defined SQL query, the user needs to execute **"Save results"** by selecting:

- **"SQL" menu** Save result

or

- icon .

This function:

- requires specifying places to save the SQL scripts saved as *.zip.

3.1.8.1.3 *History of executed SQL queries*

To check history of executed SQL queries for selected snapshot instances, the user needs to execute **"SQL queries history"** by selecting:

- **"SQL" menu** History

or

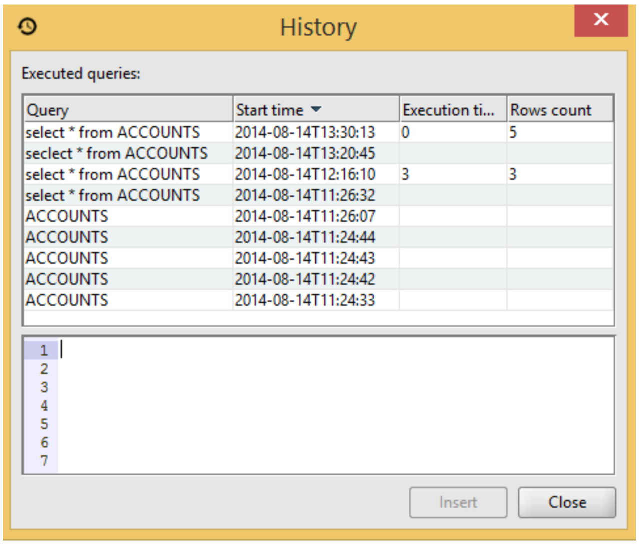
- icon

or

- keyboard shortcut: **Ctrl-H**.

This function:

- in the new window (see Figure 91) displays all SQL queries executed for the given snapshot instance,
- additionally informs the user about:
  - the period for SQL query executions,
  - time of query execution,
  - number of rows included in the SQL queries executions.



Query	Start time	Execution ti...	Rows count
select * from ACCOUNTS	2014-08-14T13:30:13	0	5
seclect * from ACCOUNTS	2014-08-14T13:20:45		
select * from ACCOUNTS	2014-08-14T12:16:10	3	3
select * from ACCOUNTS	2014-08-14T11:26:32		
ACCOUNTS	2014-08-14T11:26:07		
ACCOUNTS	2014-08-14T11:24:44		
ACCOUNTS	2014-08-14T11:24:43		
ACCOUNTS	2014-08-14T11:24:42		
ACCOUNTS	2014-08-14T11:24:33		

Figure 93. History of SQL queries

3.1.8.1.4 *Data validation in snapshot instance*

To launch validation of source data linked with snapshot instance the user needs to execute "Validate source data for marked table" in the editing mode of the chosen instance in "Snapshot instances management", by selecting:

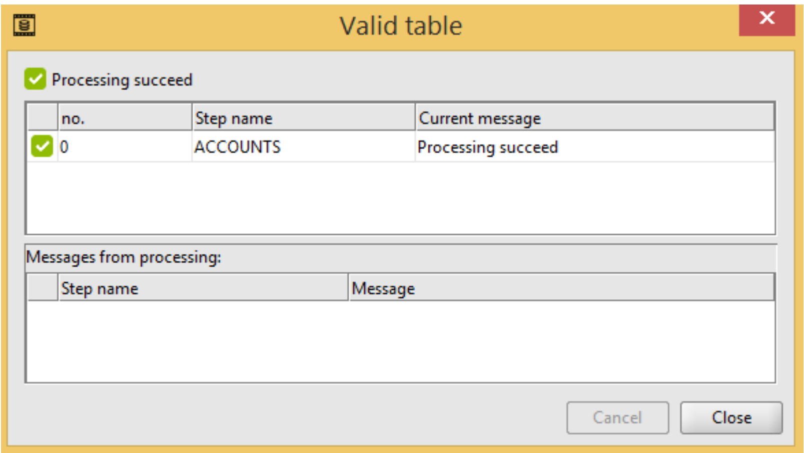
- **"Table list" menu** ✓ Validate source data for selected tables

or

- icon ✓.

This function:

- is activated only after the tables for which validation is to be performed are marked (market square: ☑),
- informs about the performed validation results,
- in case of a successfully finished action displays screen as on Figure 92,
- in case of errors informs about the errors.



✓ Processing succeed

	no.	Step name	Current message
✓	0	ACCOUNTS	Processing succeed

Messages from processing:

	Step name	Message

Figure 94. Information about the results of data validation performed in snapshot instance

3.1.8.1.5 *Load source data for the marked tables*

To import data to selected tables linked with snapshot instance or to begin ETL processing, the user needs to execute "Load source data for marked table" in the edition mode of a chosen instance in "Snapshot instances management", by selecting:

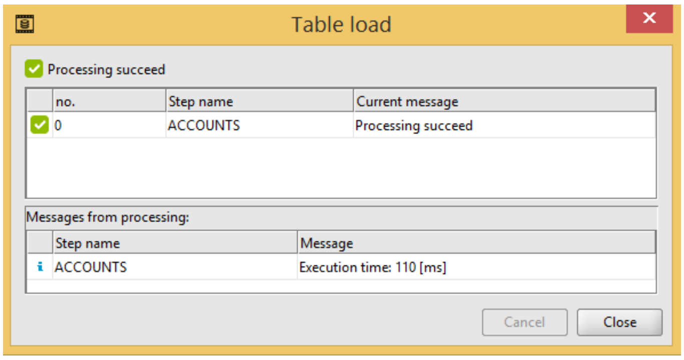
- **"Table list" menu** ⤓ Load source data for selected tables

or

- icon ⤓.

This function:

- is activated only after the tables for which data are to be imported or an optional ETL step are marked (marked square ☑),
- loads data to marked tables,
- displays the course and result of import, as on Figure 93.



☑ Processing succeed

	no.	Step name	Current message
☑	0	ACCOUNTS	Processing succeed

Messages from processing:

	Step name	Message
i	ACCOUNTS	Execution time: 110 [ms]

Figure 95. The course of data import to a snapshot instance

3.2 View

"View" menu (see Figure 94), launches the function linked with:

- refreshing information displayed on screen.

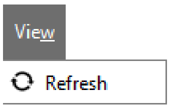


Figure 96. "View" menu in "Snapshot warehouse" module

3.2.1 Refresh

To refresh information displayed on screen the user needs to execute **"Refresh"**, by selecting:

- **View menu** ⟳ Refresh

or

- icon ⟳.

4 ETL

ETL (*Extract, Transform and Load*) – is a tool facilitating the transformation process of data included in snapshot tables.

Its task is source data transformation by:

- data joining (e.g. joining client tables with their transactions),
- data filtration (execution of appropriate data filtration in accordance with criteria expected by the supervisor),
- data verification – entering validators which allow for verification of consistency and correctness of data generated from back office systems.

To launch ETL function, select:

- **"Snapshot" menu** ETL editor

or

- icon ✎ ,

and ETL editor

or

- keyboard shortcut **Ctrl+Shift-W**.

Every time it is necessary to earlier:

- mark (highlight) the snapshot for which snapshot tables transformations are to be performed.

When ETL is activated, screen displays information as on Figure 95.

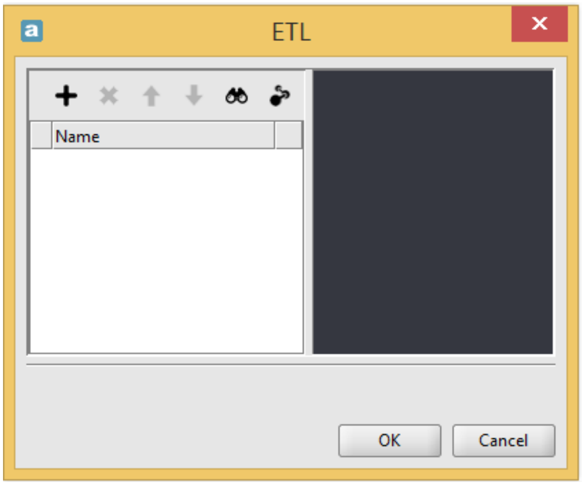


Figure 97. ETL – general view

In the upper left window panel:

- are available functions which enable:
  - appropriate filtration of data in tables,
  - execution of SQL and GROOVY scripts,
  - data validation with the use of SQL and GROOVY scripts,
  - removal of incorrectly or unnecessarily created steps,
  - change of display order in the window of created steps,
  - initial validation of created steps,
  - test execution of an SQL query,

- displayed is the list of:
  - tables defined in the selected snapshot,
  - tables (steps) created in the result of transformations performed.

In the upper part of right window panel, after selecting (marking) one of the tables displayed in the left upper window panel:

- in case of snapshot tables displayed is a column list from which the given table consists (see Figure 97),

- for tables (steps) created in the result of available functions activation, displayed are additional functions which enable execution of appropriate snapshot tables transformations (see Figure 98).

On the lower window part displayed are:

- results of executed SQL or GROOVY scripts,
- errors which occurred.

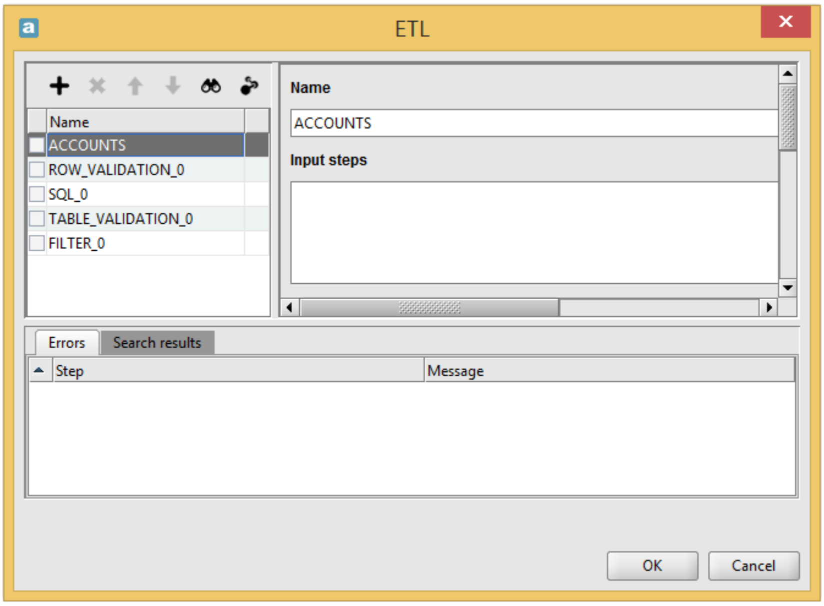


Figure 98. ETL – detailed information about columns of snapshot tables

Some of functions available here can be used always and others are activated only after the square which is located before the name of the table (step) to which they refer is marked.

4.1 Creating a new step

To create a **"New step"** the user needs to select (see Figure 97):

- icon +

and then one of available functions:

- Filtering,
- Row script,
- Table script,
- Row validation,
- SQL instruction,
- Table validation.

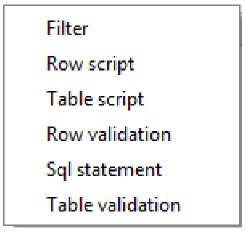


Figure 99. ETL – functions creating a new step

Selecting any function each time results in:

- creating a new step and displaying it in the left panel,
- displaying in the right window panel:
  - name of the created step,
  - name of the snapshot table for which the function is performed,
  - columns and indexes used,
  - fields to which adequate instructions are entered,
- making available functions which allow for appropriate transformation of snapshot tables.

For created steps, functions facilitating execution of table transformations are made available in the form of functional icons:

+ – add entry step / column / validator

✎ – edit column / script / validator

⧉ – copy column

★ – duplicate column

✖ – remove entry step / column / index / validator

⚙ – add index

Individual icons are shown on Figures 98 – 103 displaying screens after each function creating a new step for the table transformation is launched.

4.1.1 Filtering

Selecting **"Filter"** function by:

- icon +

and then:

- Filtering

results in:

- creating a new step which allows for data filtering in snapshot tables,
- displaying in the right window panel appropriate functions and fields as on Figure 98.

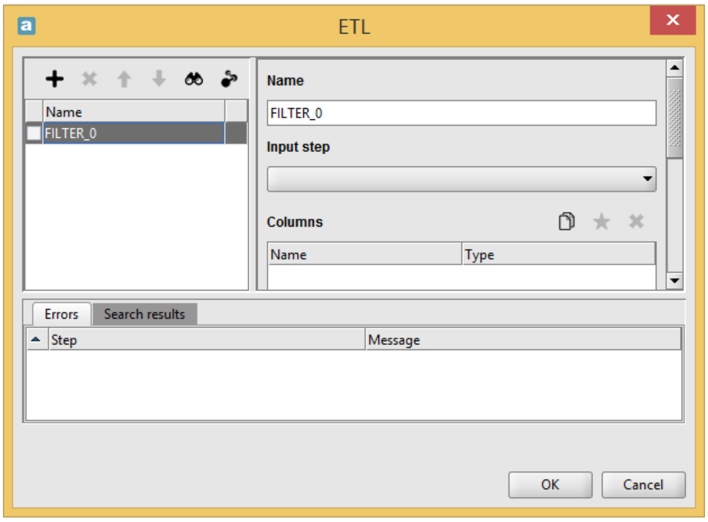


Figure 100. ETL – filtration of data in snapshot tables

4.1.2 Row script

Selecting **"Row script"** function by:

- icon +

and then:

- Row script

results in:

- creating a new step which allows for defining a script in GROOVY language which will be launched for each row,
- displaying in the right window panel appropriate functions and fields as on Figure 99.

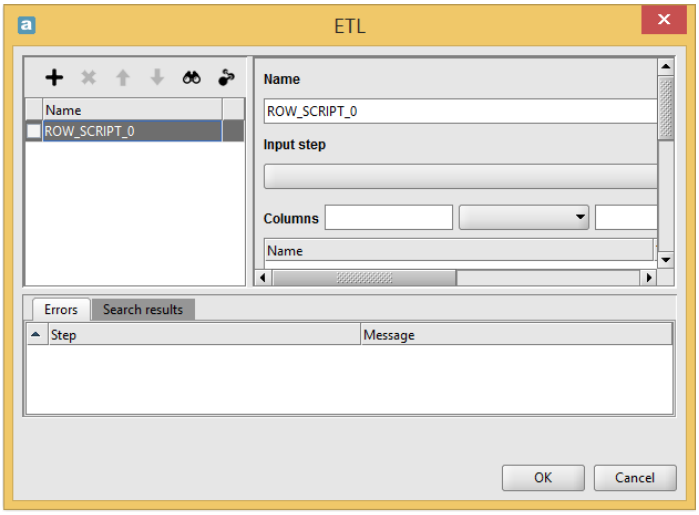


Figure 101. ETL – row script

4.1.3 Table script

Selecting **"Table script"** function by:

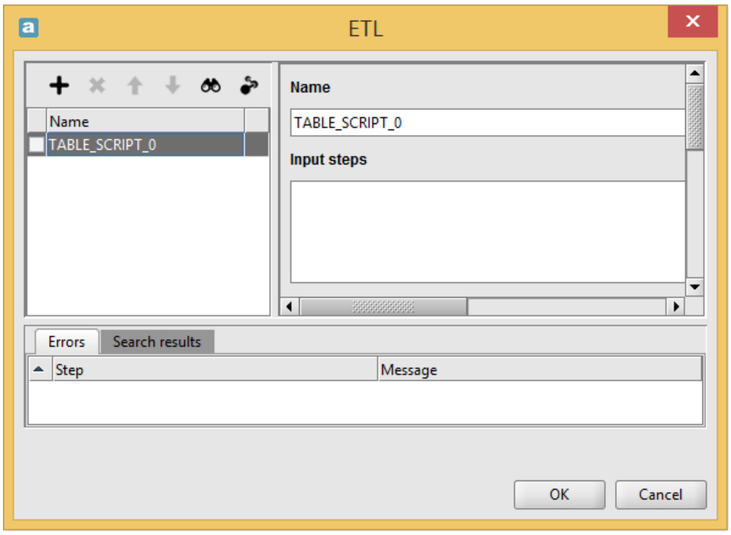
- icon +

and then:

- Table script

results in:

- creating a new step which allows for defining a script in GROOVY language in which the operation can be executed for the entire table with the use of SQL queries,
- displaying in the right window panel appropriate functions and fields as on Figure 100.



Name: TABLE_SCRIPT_0

Name
TABLE_SCRIPT_0

Figure 102. ETL – table script

4.1.4 Row validation

Selecting **"Row validation"** function by:

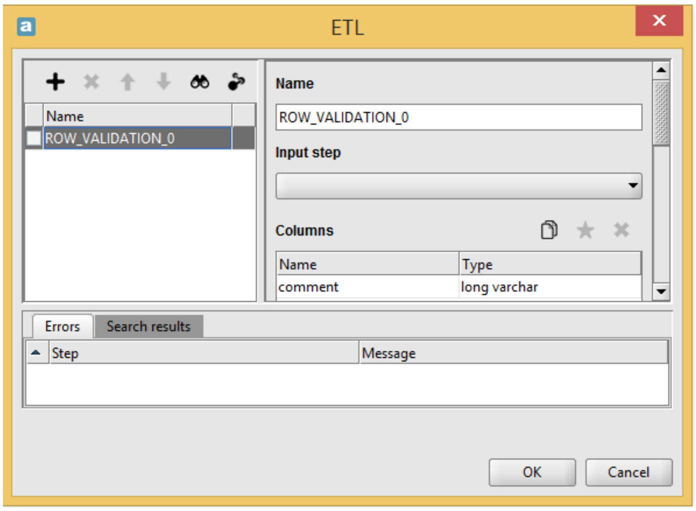
- icon +

and then:

- Row validation

results in:

- creating a new step which allows for defining single rows with data using SQL and GROOVY scripts,
- displaying in the right window panel appropriate fields and functions as on Figure 101, which allow for defining an appropriate validation.



Name: ROW_VALIDATION_0

Input step:

Columns

Name	Type
comment	long varchar

Step	Message

Figure 103. ETL – data validation with the use of SQL and GROOVY scripts

4.1.5 SQL instruction

Selecting **"SQL instruction"** function by:

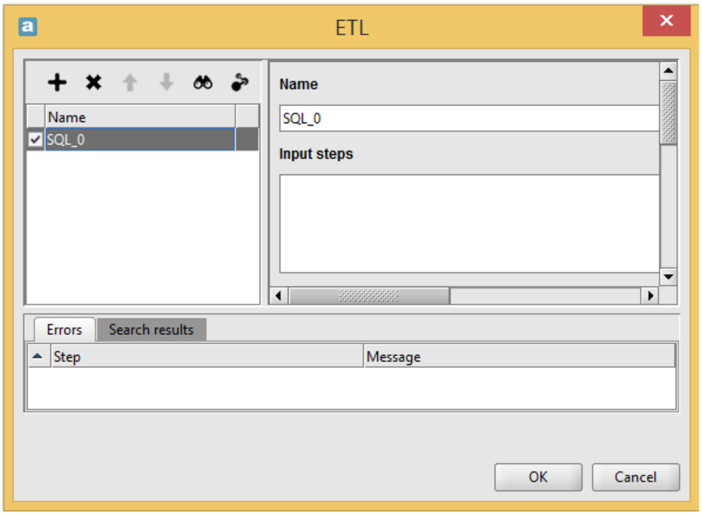
- icon +

and then:

- SQL instruction

results in:

- creating a new step which allows for defining an SQL query which will be performed on entry tables; the result will be located in the exit table,
- displaying in the right window panel appropriate fields and functions as on Figure 102, which enable defining an appropriate query.



Name: SQL_0

Input steps

Step	Message

Figure 104. ETL – SQL instruction

4.1.6 Table validation

Selecting **"Table validation"** function by:

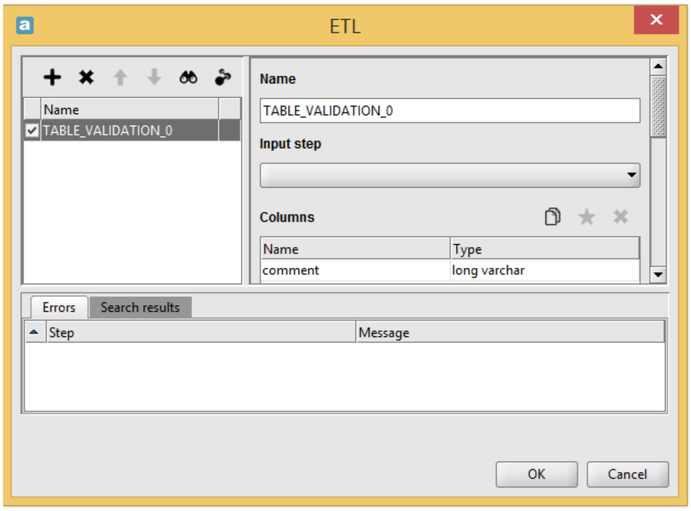
- icon ✚

and then:

- Table validation

results in:

- creating a new step which allows for validation of the entire table with data using SQL and GROOVY scripts,
- displaying in the right window panel appropriate fields and functions as on Figure 103, which allow for defining an appropriate validation.



Name
☑ TABLE_VALIDATION_0

Name: TABLE_VALIDATION_0

Input step:

Columns

Name	Type
comment	long varchar

Step	Message

Figure 105. ETL – table validation

4.2 Removing a step

To remove an incorrect or unnecessary step the user needs to:

- mark the square located before the name of the removed step (for exampl ☑ TABLE_SCRIPT_0),
- with the functional icon ✖ located above the list of tables (steps) select the **"Remove step"** function.

This function:

- removes irreversibly the marked step,
- refers only to those steps (tables) created as the result of the performed transformations,
- is not active for tables defined in the selected snapshot.

4.3 Change of display order of defined steps

To change the order of displayed defined steps on the list, the user needs to:

- mark the square located before the name of the step to be moved (for example: ☑ TABLE_SCRIPT_0),
- with one of the functional icons ↑, ↓, available above the list of tables (steps), select the **"Change order of step execution to earlier/later"** function.

These functions:

- by setting the display order of tables on the list, make it possible to set the required order of individual steps execution,
- apply to all tables displayed on the list.

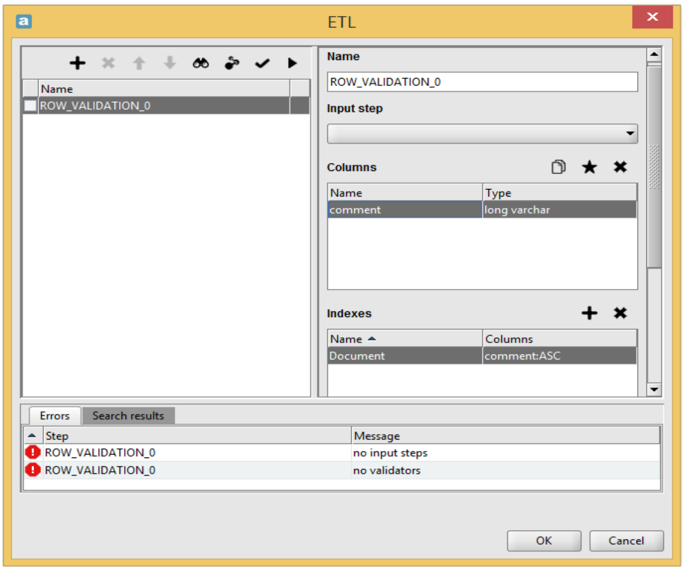
4.4 Validation of defined steps

To perform validation of defined steps the user needs to use "Validate steps" function by selecting:

- Icon ✓ available above the list of tables (steps),

This function:

- can be launched at any time in the process of step creation,
- results of validation are displayed in the lower part of the screen, as on Figure 104.



Name
□ ROW_VALIDATION_0

Name: ROW_VALIDATION_0

Columns

Name	Type
comment	long varchar

Indexes

Name	Columns
Document	comment:ASC

Step	Message
ROW_VALIDATION_0	no input steps
ROW_VALIDATION_0	no validators

Figure 106. ETL – step validation results

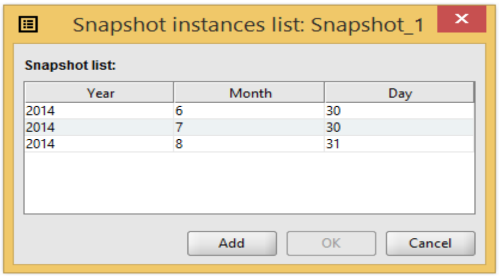
4.5 Test execution of defined steps

To perform a test verification of defined steps the user need to select **"Test execution of steps"** by choosing:

- Icon ▶ available above the list of tables (steps).

This function:

- can be performed at any time in the process of step creation,
- by moving to "Snapshot instances management" allows for test execution of defined steps,
- displays the screen as after launch of "Manage snapshot instances" function (see Figure 105 and compare with Figure 85).



Snapshot instances list: Snapshot_1

Snapshot list:

Year	Month	Day
2014	6	30
2014	7	30
2014	8	31

Figure 107. ETL – list of snapshot instances available at test execution of defined steps

After selecting (highlighting) the required snapshot instance for which tests are to be performed and confirming the choice with the OK button, all functions available from the "Snapshot instances management" level are activated (see chapter 3.1.8 Manage snapshot instances).

5 Fetching by mappings

To launch the "Fetching by mappings" function the user needs to select the Reports bookmark, and then:

- **Report Menu** Fetching data by mappings

This function:

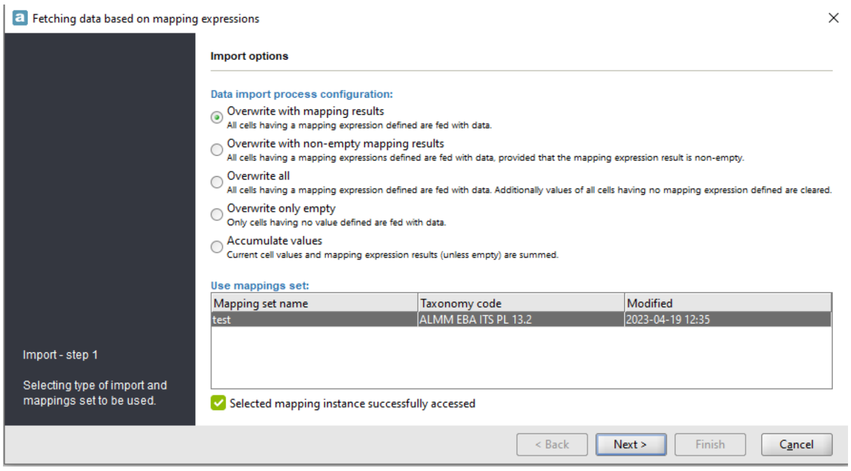
- is available only for banks owning an extended license including "Data mapping" and "Snapshot warehouse" modules,

- allows for data fetching to aSISt against a mapping expressions specified for individual reporting cells and source to download,
- applies always to one active report,
- can be executed simultaneously only on one workstation,
- is active only when none of the users work in the "Data mapping" or "Snapshot warehouse" modules.

When "Fetching by mappings" function is selected, the screen displays information as on Figure 105.

The full "Data fetching by mappings" process is executed in **six steps**.

Step 1 – selection of the import type and mappings set (see Figure 108).



Fetching data based on mapping expressions

Import options

Data import process configuration:

● Overwrite with mapping results
All cells having a mapping expression defined are fed with data.

○ Overwrite with non-empty mapping results
All cells having a mapping expressions defined are fed with data, provided that the mapping expression result is non-empty.

○ Overwrite all
All cells having a mapping expression defined are fed with data. Additionally values of all cells having no mapping expression defined are cleared.

○ Overwrite only empty
Only cells having no value defined are fed with data.

○ Accumulate values
Current cell values and mapping expression results (unless empty) are summed.

Use mappings set:

Mapping set name	Taxonomy code	Modified
test	ALMM EBA ITS PL 13.2	2023-04-19 12:35

✓ Selected mapping instance successfully accessed

Import - step 1

Selecting type of import and mappings set to be used.

Figure 108. Import options in "Data fetching"

Import options are the same as in "Import data" function, i.e.:

Overwrite with mapping results – will result in:

- **import of all data against a defined expression and data sources,**
- retaining only these previously entered data that does have mapping expressions defined.

Overwrite with non-empty mapping results – will result in:

- **import of all data against a defined expression and data sources for which the mapping result is not empty,**
- retaining only these previously entered data that does not have mapping expressions defined or the mapping result was empty.

Overwrite all – will result in:

- **import of all data against a defined expression and data sources,**

- removal of all data which were entered before the import (applies to those reports to which data from external files will be imported).

Overwrite only empty – will result in:

- **import of only those data which in the reporting period did not have any values entered,**
- retaining of all data entered before import.

Accumulate values – will result in:

- **import of all data against a defined expression and data sources,**
- values from external files and previously entered will be summed up.

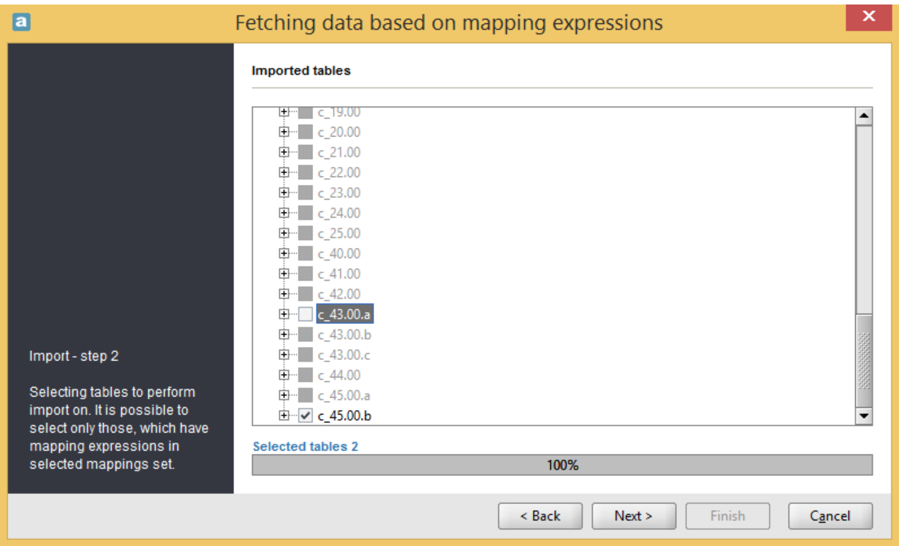
It is always possible to select one of these options.

In the lower part of the screen a list of all available sets with mapping expressions is displayed and it is necessary to specify one of them.

The following elements only are included in this list:

- sets applying to reports currently subject to data fetching,
- sets with "active" status (deactivated sets for mapping are invisible here).

Step 2 – choosing tables to import (see Figure 109).



Fetching data based on mapping expressions

Import - step 2

Selecting tables to perform import on. It is possible to select only those, which have mapping expressions in selected mappings set.

Imported tables

- ▨ c_19.00
- ▨ c_20.00
- ▨ c_21.00
- ▨ c_22.00
- ▨ c_23.00
- ▨ c_24.00
- ▨ c_25.00
- ▨ c_40.00
- ▨ c_41.00
- ▨ c_42.00
- □ c_43.00.a
- ▨ c_43.00.b
- ▨ c_43.00.c
- ▨ c_44.00
- ▨ c_45.00.a
- ☑ c_45.00.b

Selected tables 2

100%

Figure 109. Choosing tables to "Fetching by mappings"

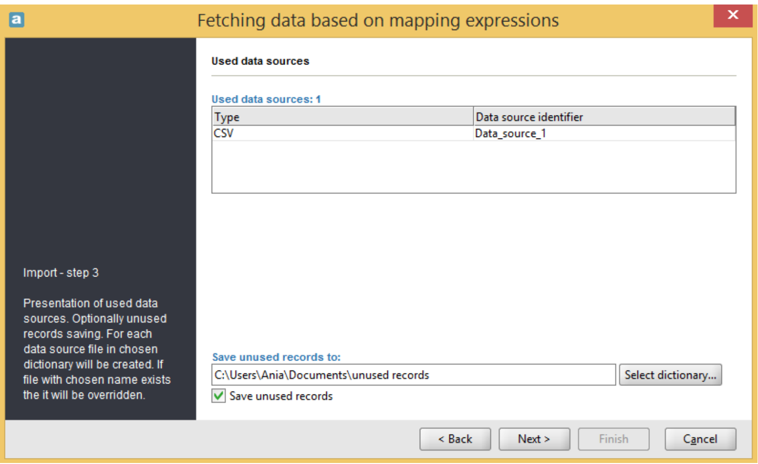
Data fetching is to be performed for tables displayed here in the tree form.

☑ – means that the given table will be subject to data fetching (Figure 109: c_45.00.b),

□ – means that the user aborted data fetching for the table located alongside (Figure 109: c_43_00.a),

■ – means that for this table in the set specified in step 1 there are no defined mapping expressions and the data fetching by expressions will not be performed for the table.

Step 3 – displaying used data sources (see Figure 110 and 111).



Fetching data based on mapping expressions

Used data sources

Used data sources: 1

Type	Data source identifier
CSV	Data_source_1

Import - step 3

Presentation of used data sources. Optionally unused records saving. For each data source file in chosen dictionary will be created. If file with chosen name exists the it will be overridden.

Save unused records to: C:\Users\Ania\Documents\unused records

☑ Save unused records

Figure 110. Displaying used data source and selection of "unused records tracking" option

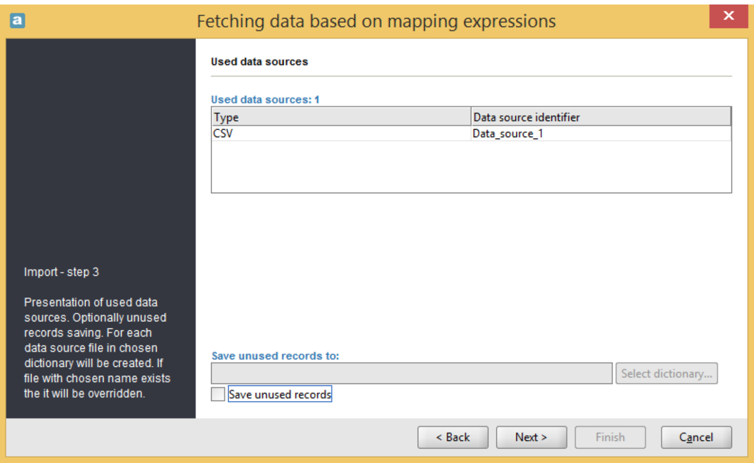
In this step information about used data sources is displayed.

Additionally, it is possible here to mark the option allowing for display of unused records list in step six.

Marking position ☑ Save unused records (see Figure 110) activates button Select dictionary... and allows to specify a place to save the file with unused records (a separate file for each data source).

If option □ Save unused records is not marked (see Figure 111) then:

- the option "unused records tracking" will not be used,
- there will be no information regarding whether all records were used in the process of data fetching.



Fetching data based on mapping expressions

Used data sources

Used data sources: 1

Type	Data source identifier
CSV	Data_source_1

Save unused records to:

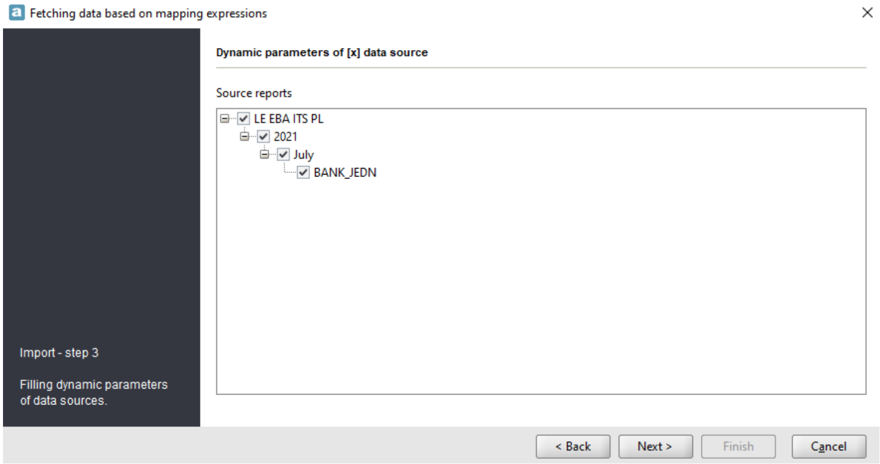
□ Save unused records

Import - step 3

Presentation of used data sources. Optionally unused records saving. For each data source file in chosen dictionary will be created. If file with chosen name exists the it will be overridden.

Figure 111. Displaying used data sources

For a relative period report, the data fetch wizard will point to the data based on the 'Relative period' field available when creating and editing the data source (see chapter: 2.2.1.1.2 Inner data source – source reports type.).



Fetching data based on mapping expressions

Dynamic parameters of [x] data source

Source reports

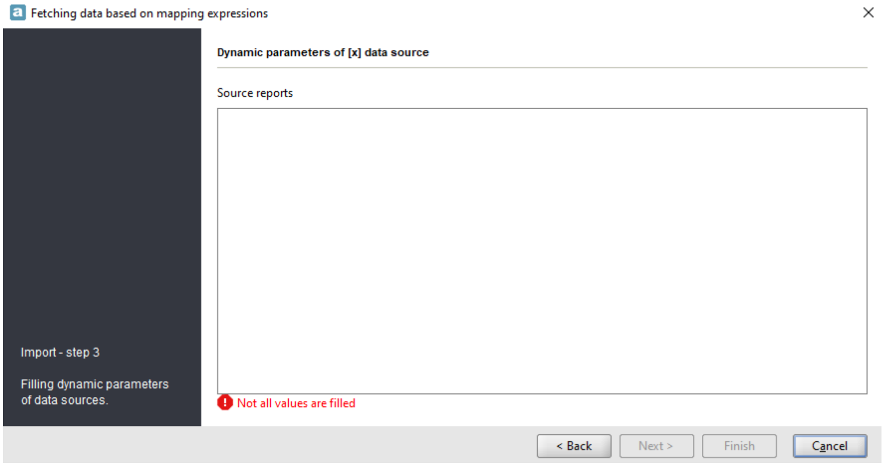
- ☑ LE EBA ITS PL
  - ☑ 2021
    - ☑ July
      - ☑ BANK_JEDN

Import - step 3

Filling dynamic parameters of data sources.

Figure 112. Dynamic parameters of data source

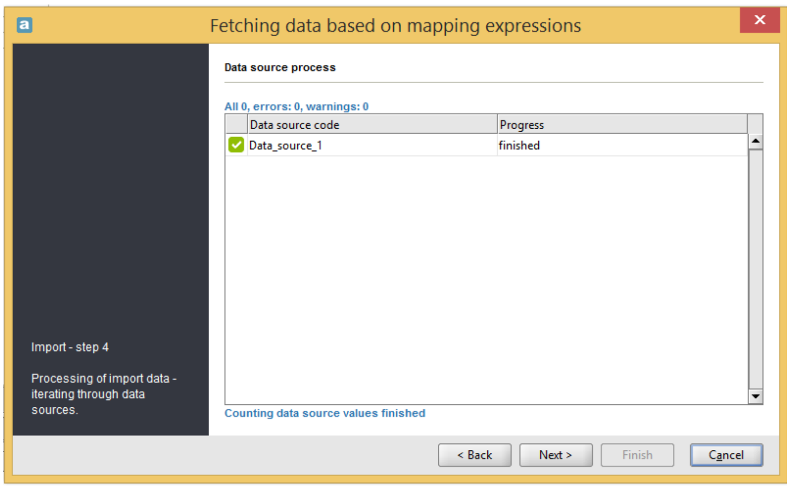
If the report is missing for the period indicated (e.g. for a value of 1 M it will be the previous month), a warning message will be displayed:


Fetching data based on mapping expressions
Dynamic parameters of [x] data source
Source reports
Import - step 3
Filling dynamic parameters of data sources.
< Back
Next >
Finish
Cancel

Not all values are filled

Figure 113. Dynamic parameters of data source – not all values are filled

Step 4 – using data to import (see Figure 110).


Fetching data based on mapping expressions
Data source process
Import - step 4
Processing of import data - iterating through data sources.
< Back
Next >
Finish
Cancel

All 0, errors: 0, warnings: 0

Data source code	Progress
✓ Data_source_1	finished

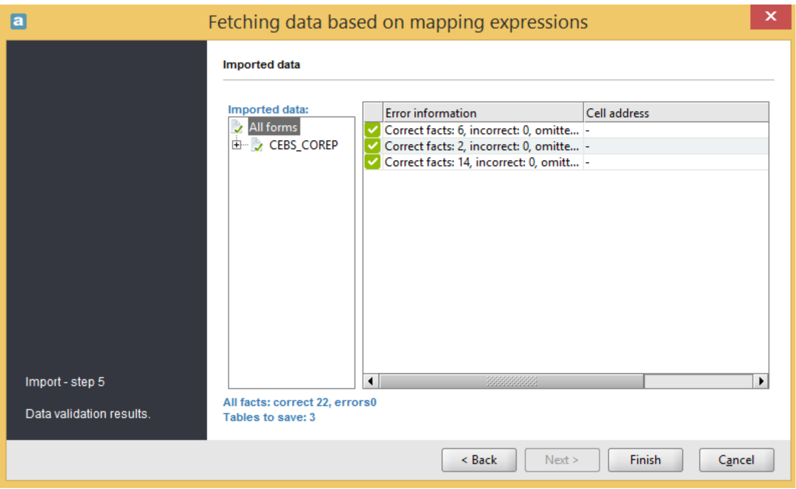
Counting data source values finished

Figure 114. Data fetching by mappings – using data to import

The following elements are displayed in this window:

- information about errors detected during validation,
- data source code on the basis of which data import by mappings is executed.

Step 5 – displaying results of the import performed (see Figure 115):


Fetching data based on mapping expressions

Imported data

Imported data:

- All forms
  - CEBS_COREP

	Error information	Cell address
✓	Correct facts: 6, incorrect: 0, omitte...	-
✓	Correct facts: 2, incorrect: 0, omitte...	-
✓	Correct facts: 14, incorrect: 0, omitt...	-

All facts: correct 22, errors0
Tables to save: 3

Import - step 5

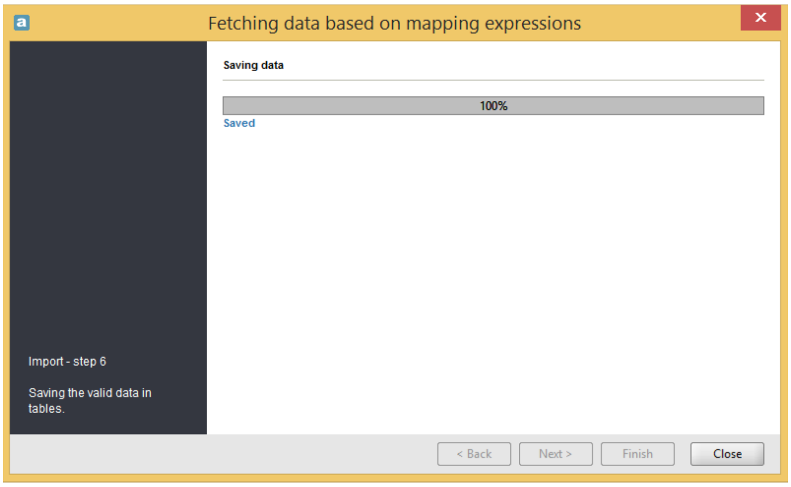
Data validation results.

Figure 115. Displaying validation results

Detailed results of the validation performed refer in this case to:

- the highlighted list of tables to which data can be imported and,
- information about errors detected during validation.

Step 6 – saving correct data to individual tables (see Figure 116).


Fetching data based on mapping expressions

Saving data

100%

Saved

Import - step 6

Saving the valid data in tables.

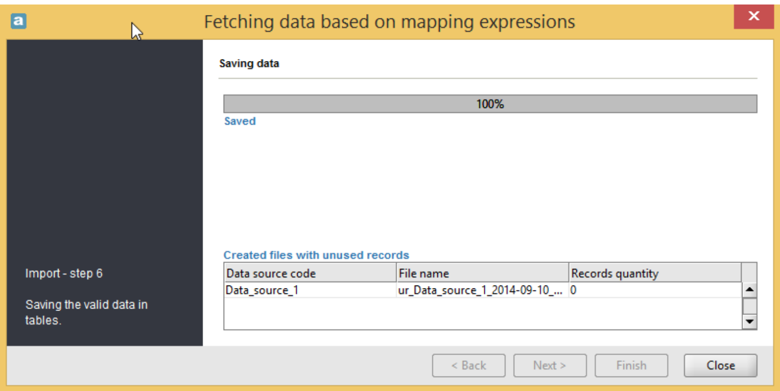
Figure 116. Saving correct data to tables

Saving of correct data to tables with defined mapping expressions is performed here. Incorrect data are always ignored.

If in step 3 the option of tracking unused records was marked then additionally in the lower part of the screen (see Figure 110) information about created files with such unused records is displayed in the table form, with details concerning:

- data source code,

- name of the created file,
- number of unused records saved to this file.



Fetching data based on mapping expressions

Saving data

100%

Saved

Import - step 6

Saving the valid data in tables.

Created files with unused records

Data source code	File name	Records quantity
Data_source_1	ur_Data_source_1_2014-09-10_...	0

Figure 117. Saving correct data and unused records

This table is empty because all data were correct.

6 Additional information

6.1 Data sources

Data sources are the sources referred to by defined data mapping expressions.

They are defined globally for the entire application and shared by the expressions which use them.

They have a unique identifier, further referred to as a data source code.

Tabular data source – describes data displayed in a correct table, where:

- the first row represents table headings, and the remaining rows represent data,
- each column has a unique heading (which is a correct identifier compliant with the following format:
[a-zA-Z][a-zA-Z0-9_]*,
- all columns have the same number of rows.

6.1.1 Excel type tabular data sources

Properly defined Excel type source:

- has a path to the file compliant with Excel 2003 format,
- has a spreadsheet number (spreadsheets are numbered beginning with 1),
- has correct tabular data beginning in the first column and the first row of a given spreadsheet,
- can have empty rows – they will be omitted at file upload.

6.1.2 CSV type tabular data sources

Properly defined source:

- has a path to the file compliant with CSV format,
- has column names in the first row of the file,
- has fields separated with one of the selectable separators.

6.2 Mapping expressions

Mapping expressions are the formulas referring to the defined data sources in order to draw the values from them and optionally perform additional tasks (such as e.g. a sum of numerical values).

Except for the below described specific situations, a single expression can use any amount of source data.

Data sources need to be previously defined in order to be specified in a mapping expression.

6.2.1 Types of reporting cells

While creating **mapping expressions, four types of formulated expressions are permitted.** These types correspond to mapped cells types:

- NUMERICAL:

Numerical expression consists of any arithmetic operations.

Permitted operations: **floating-point division and multiplication, addition, subtraction** and nesting in brackets.

Numerical expressions can refer to any number of calculated values in data sources.

Numerical constants with a full stop as a decimal separator can also be used.

<arithmetic expression> ::= (<arithmetic argument> | <arithmetic operation>)

<arithmetic argument> ::= (<number> | <data source value>)

<arithmetic operation> ::= (<arithmetic operation><arithmetic operator><arithmetic argument> | "("<arithmetic operation> ")" | "-" <arithmetic operation>)

<arithmetic operator> ::= ("/" | "*" | "+" | "-")

- LOGICAL:

Logical expression – any expression using logical data source values, logical operators and logical constants (**true**, **false**).

Logical operators (<&&> i <||>) denote conjunction and logical alternative respectively, where conjunction has a higher priority.

Nesting in brackets is also possible. Expressions in brackets have the highest priority.

<logical expression> ::= (<logical argument> | <logical operation>)

<logical argument> ::= (<logical constant> | <data source value>)

<logical operation> ::= (<logical operation><logical operator><logical argument> | "("<logical operation > ")")

<logical operator> ::= ("&&" | "||")

<logical constant> literal **true** and **false**, all letter cases are permitted.

- TEXTUAL:

Textual expression – a textual constant or textual data source value.

<textual argument> ::= (<textual constant> | <data source value>)

<textual constant> any character string enclosed in a single quotation mark.

" \ " is an escape character in textual literals, so in order to input **"’"** in the textual content, it needs to be preceded with an escape character.

All other characters preceded with this character will be entered without change.

Simultaneously, the content of a correct text cannot end with either of both special characters.

Quotation marks can be omitted in texts which are not keywords, begin with the letter and have no white characters.

- DATE:

Date type expression can only be a value of one data source of the same type.

<date expression> ::= <data source expression>

Expression type is selected automatically (according to the type included in the taxonomy) and becomes a basis for control of the edited expression type.

All data sources occurring in it also need to return a value in accordance with the type specified for the entire expression.

All above expressions use the value description downloaded from the source in accordance with the below syntax:

<source_identifier>:{ <expression> }

<source_identifier> a unique data source identifier (provided at creation) – must have the format of a correct identifier described by a regular expression [a-zA-Z][a-zA-Z0-9_]*

<expression> described further in this chapter, calculates a value returned by a specified data source.

For example:

ds:{ 1 }
where 'ds' is an identifier and '1' is an expression.

Expression type is selected automatically (according to the type included in the taxonomy) and becomes a basis for control of the edited expression type.

Most often expressions mapping reporting data include **numerical values.**

In such cases a mapping value uses numerical values and available for them operators:

<**+**> – denoting addition,
<**-**> – denoting subtraction,
<*> – denoting multiplication,
<**/**> – denoting division.

These operators are joined in accordance with the applicable rules – multiplication and division before addition and subtraction.

In a case of more complex calculations, in order to force a correct argument joining in the operation, bracket operators <(> and <)> can be used. Hence the formula:

2 * (2 + 2), when calculated, returns 8.

However, some reporting cells contain other types of data.

Some cells occurring in aSISt are treated in the mappings module as **textual values**:

- editable from the list of selected values (e.g. currencies) or
- entered directly (e.g. unit name).

In case of such cells, a value can be drawn only directly from the data source (arithmetic operations are not permitted) or use a textual constant.

Reporting cell can also be of a **date type** (in this case the calendar in aSISt can be used for editing) – then only a single data source from which the date will be drawn can be used for mapping.

6.2.2 Types of mapping expressions

There are three types of mapping expressions:

- **Single reporting cell mapping expressions** (not member of a tuple) – such cell can have a maximum of one mapping expression referring to any number of defined data sources.

- **Mapping expression of table list cells or typed dimensions.** This expression resembles the mapping expression of a common reporting cell, however, the number of values reached in the course of its calculation is equal to the number of list records or, respectively, to a number of value of typed dimensions obtained in the fetching process.

- **Grouping expressions**.

An expression entered into a special field for list tables or at defining typed dimensions which at fetching by mapping process specified the number of list records or typed dimension's values obtained. This expression must refer to one data source only and use one of the available grouping functions.

A detailed description of list tables mapping can be found further in this chapter.

6.2.2.1 Mapping of single reporting cells

In the easiest case:

- mapping expressions refer to one data source and
- the value calculated by the data source is recorded in the mapped cell.

In such case, a mapping expression has the following form:

ds{ <aggregate expression> }

where:
ds is a data source identifier
aggregate expression chooses required data source records and performs on them necessary calculations.

For example:
Adding up account balances in the required currency for a data source with the data symbol, with columns named balance and currency, can look as follows:

data:{ sum($balance, $currency = PLN) }

To feed the average balance sum in PLN from two different sources data1 and data2 into the cell, mapping is required:

(data1:{ sum($balance, $currency = PLN) } + data2:{ sum($balance, $currency = PLN) }) / 2

6.2.2.2 Using data from the source

Data sources are table data sets in which every column has a unique name.

In mapping expressions, in order to refer to a value in the given column, use its name preceded with <$> sign

For example: *$balance_owed*

Source data expression (part of the expression between <{> and <}>) tries to calculate a single value on the basis of a specified number of records from the source.

For this reason aggregate functions are used. Each of the available aggregate functions calculate one value.

All aggregate functions can be schematically presented as follows:

<function_name>(<expression>, <discriminator>)

<expression> is in general any expression which is a basis for calculating an aggregate function. Most often it is a single source data column (e.g. $balance).

For example, a search of the highest balance can have the following form:
max($balance)

Often, however, within the interest of a given aggregate function, there is only a part of set records. For this purpose an additional condition (**<discriminator>**) can be added.

For example:
max($balance, $currency = PLN)
will perform a search for the highest balance only in rows which in the column $currency have the PLN value.

In more complex cases it can turn out that a single condition is not sufficient. In cases in which summing up a column $balance_owns for PLN and those records which in column $balance_owed have the value bigger than one is required. Then the following formula is needed:

sum($balance_owns, $balance_owed > 0 && $currency = PLN)

In this case a logical operator was used which checks both conditions simultaneously (conjunction). The mechanism of construing complex conditions is described further in this chapter.

Available **aggregate functions** sum, avg, first, last, min, max, count, same, one, any are described in details in the chapter 6.2.6.6 6.2.6.6 Aggregate function.

6.2.2.3 Mapping of list cells

Mapping expressions in **list tables** (tuple type) have a similar form to the above-mentioned, however, there are certain differences connected with the way of their interpretation.

The nature of list tables suggests that the number of list elements is not known – it depends, for example, on the number and types of transaction in a given reporting period.

For this reason it is necessary to calculate all values (one for each list table record) using one mapping expression.

For this operation to be possible, it is necessary to define a grouping expression.

A grouping expression in case of list tables is located in the first column (first row – depending on the table orientation).

In the report such cell is uneditable.

A grouping expression **has to use exactly one data source** and has the following form:

def:{ grouping_function(<arguments>) }

Grouping functions **do not calculate values but define row groups** which serve as the basis to set values of other expressions in the group.

In case of list tables a group consists of all expressions within one list record.

There are two grouping functions:

- **group** – grouping, accepts a non-empty list of grouping expressions of any type.

Each argument value vector specifies a group.

- **conditionGroup** – conditional grouping.

As a first argument accepted is the condition met by all groups set out by the function. Subsequent arguments are a list of grouping expressions of any type.

Each argument value vector specifies a group.

6.2.2.4 Mapping of typed dimensions

In case of typed dimensions mapping it is necessary to define typed dimensions values which will be calculated in the fetching process.

Similarly as in mapping of list cells, here it is also **necessary to define grouping expressions, calculating as many values of result cells, as many different values of typed dimension were calculated.**

For this purpose two grouping functions are used:

- **group** – grouping of defined-type expressions,
- **conditionGroup** – conditional grouping.

6.2.3 Verification of mapping expressions correctness

Built mapping expressions are subject each time to a verification process which checks whether:

- they comply with the binding syntax,
- they are correct with respect to the reporting cell type,
- all data source identifiers used are correct (there exist such data sources).

In case the system finds an irregularity, the user is informed each time about it by an appropriate error message.

If the editing process is finished correctly, automatic formatting of expressions entered takes place, that is:

- entered text is aligned,
- surplus brackets are deleted,
- multiple nesting of '-' or '+' operator before numerical literals is removed,
- the second argument of the aggregate function is removed if the function is a literal and has a true value,
- a formal entry of same function is removed if it does not contain additional conditions and is not nested in any other aggregate function,

- surplus to_number projection occurrence is removed (in case the expression type is unambiguous),
- date format is removed from the to_date projection when it has a default format ('yyyy-MM-dd').

6.2.4 Language syntax of grouping expressions

A grouping expression specifies exactly one data source containing a grouping expression.

<grouping expression> ::= <data source code>":" "{" <data source grouping expression> "}"

Grouping expressions do not calculate cell values but specify value groups which are the basis for valuation of other source expressions.

Expressions dependent on a grouping expression generate a separate value set for each group specified by this expression.

This property is used in mapping of typed dimensions and tuples in order to create multiple value groups for one expression group.

6.2.5 Calculating the value of a mapping expression

Calculating the value of a mapping expression requires calculation of all values in used data source expressions.

Each of the data source used has to describe exactly one value.

One expression can contain any number of expressions using different data sources, and the data source can be used to feed multiple values.

An error returned by at least one used data source causes an error in calculation of the entire expression.

In case of expressions enclosed in a grouping expression, all subexpressions using the same data source as the grouping expression are a basis to generate multiple values for one mapping expression.

Valuation of such expression takes place multiple times – for each group specified by the grouping expression.

6.2.6 Language of table data sources

6.2.6.1 Grammar

Tabular language of data source expressions (Excel and CSV) supports two expression types:

- **"tabular expressions"** calculating values and
- **"grouping tabular expressions"** specifying groups which are the basis for valuation of other tabular expressions.

Language defines expressions compliant with the grammar:

<tabular expression> ::= <operation>

<operation> ::= (<argument> | <arithmetic operation> | <comparison operation> | "(" <operation> ")")

<arithmetic operation> ::= (<operation><arithmetic operator><argument> | <operation><logical operator><argument>)

<arithmetic operator> ::= ("+" | "-" | "/" | "*")

<logical operator> ::= ("||" | "&&")

<comparison operation> ::= (<operation><comparison operator><operation> | <operation><comparison operator><set> | <set><comparison operator><operation>)

<comparison operator> ::= ("=" | "!=" | "<" | "<=" | ">" | ">=")

<set> ::= ("[" "]" | "[" <operation> ["," <operation>]* "]")

<argument> ::= (<column> | <literal> | <function> | "*")

<literal> ::= (<numeral constant> | <logical constant> | <textual constant>)

<function> ::= <function name>"(" [<function arguments>] ")"

<function arguments> ::= <operation> ["," <operation>]*

<numerical constant> – any numerical constant with **"."** as a decimal separator

<logical constant> – true or **false** – any letter case

<textual constant> – any string of alphanumeric characters without white signs, not beginning with a number, or any string of characters in single quotation marks (**"’"**). An escape character is **"\ "**. It allows for the use of special characters as common characters in brackets. The use of an escape character before non-special characters is ignored.

<column> a correct column identifier preceded with **"$"**

<function name> one of the accepted function names operating on values, aggregate or double aggregate

<grouping tabular expressions> ::= <grouping function name>"(" [<function arguments>] ")"

<grouping function name> one of the accepted grouping function names

6.2.6.2 Operations

As mentioned earlier, it is possible to build complex conditions limiting selected records in aggregate functions. It can be achieved with the use of the following mechanisms:

- **logical operators,** sum and conjunction, respectively, which require the following logical arguments**:**

&& – denoting conjunction **"and"**

|| – denoting alternative **"or"**

for example:

currency equal to PLN and EUR chooses a following condition:

$currency = PLN **&&** *$currency = EUR*

currency equal to USD or EUR chooses a following condition:

$currency = USD || $currency = EUR

simultaneous addition of a condition to the balance looks as follows:

(currency = USD || $currency = EUR) && $balance > 0

and chooses the USD or EUR currency the balance of which is greater than zero

- the following are **number comparison operators**:

= denoting: **equal**
!= denoting: **unequal**
< denoting: **less than**
<= denoting: **less than or equal**
> denoting: **greater than**
>= denoting: **greater than or equal**

- **comparison operators for textual values:**

= denoting: **equal**
!= denoting: **unequal**

Textual values can be compared with the use of a regular expression (described further in this chapter).

Operator 'equal' and 'unequal' is performed on an argument of any type, and the remaining comparison operators support numerical and date types.

- **comparison with the use of a set operator: [<element1>, <element2>, ...]** (described further in this chapter).

A discriminator can have any length and use multiple columns – in order to retain the correct condition it can require the use of brackets.

Operations are grouped according to priorities (increasing):

- **"||"**
- **"&&"**
- **"=", "!=", "<", "<=", ">", ">="**
- **"+", "-"**
- **"*", "/"**

6.2.6.3 Set operator

Set operator can occur only on one side of a comparison and support only the following operations:

= denotes **a member** of the set
!= denotes **a non-member** of the set

the elements of which are listed between [ ] brackets and separated with a comma.

Equality operation (set theoretical: 'is a member of') is semantically equal to the alternative of value comparison with each element of the set. Simultaneously, a comparison with an empty set is always false.

Inequality operation (set theoretical: 'is not a member of') is semantically equal to a comparison conjunction (operator **"!="**) with each element of the set. Simultaneously, a comparison with an empty set is always true.

This operator is used in simplified notations of long conditions with a regular form (for example listing any permitted value for each column).

For example:

sum($balance, $currency = PLN || $currency = EUR || $currency = USD || $currency = GBP)

is equal to:

sum($balance , $currency = [PLN, EUR, USD, GBP])

6.2.6.4 Value manipulating functions

- **toNumber** – explicit conversion to a floating point numerical type. Conversion argument has a string type. A second, optional, textual argument is a format which in case of the numerical type is a one-character decimal separator. In case the argument is absent, a default ('.') is selected.

The function is required when:

○ values in data source have an other than required (full stop) decimal separator. In case of a comma, the formula looks as follows:

sum(toNumber($balance, ','))

○ it is uncertain whether textual or rather numerical values should be compared (it cannot be deduced from the comparison itself – the values in this case are treated as text)

avg($balance1, toNumber($balance1) = toNumber($balance2))

- **toBool** – explicit conversion to a logical type. Conversion argument has a string type. A second, optional, textual argument is a format which in case of the logical type looks as follows: '<value-true>|<value-false>'. In case the argument is absent, a default ('true|false') is used,

for example:

sum($balance, toBool($interesting, 'YES|NO'))

- **toDate** – explicit conversion to a date type. Conversion argument has a string type. Other optional string argument is the date format. In case the argument is absent, a default ('yyyy-MM-dd') is selected.

Dates used as mapping values are returned to the cell:

toDate($transaction_date)

or in case the date has a format other than default:

toDate($transaction_date, 'dd.MM.yyyy') for date of '1.1.2009' type

it also applies to comparisons when, for example, the result to obtain is a sum of transaction values since the year beginning:

sum($value, toDate($transaction_date) >= toDate('2009-1-1'))

- **toString** – a function opposite to explicit conversions – creates text from a given type. First argument is a value processed to text, the second one is an optional format compliant with those from the conversion. In case the second argument is absent, a default format is used.

- **abs** – absolute value. A single-operand function, accepts and returns values of numerical type.

Any arithmetic expression or a single column can be used as an argument.

For example:
the number of records with balances over the threshold value can be presented as follows:
*count(*, abs($balance_owns- $balance_owed) > 10000)*

Absolute value can also be used in case it is uncertain which sign (positive or negative) occurs with the data in the data source:

sum($balance_owns – abs($balance_owed))

- **negate** – an opposite value. A single-operand function, accepts and returns values of numerical type.

- **roundInt** – rounds a floating point value to an integer according to general rules of rounding. A single-operand function, accepts and returns values of numerical type.

- **toInt** – projection on the integer. In case a number has a non-zero decimal expansion, an error will occur. A single-operand function, accepts and returns values of numerical type.

- **isInt** – checks if the string is a representation of an integer. A single-operand function, takes a textual type argument and returns a logical value.

- **toUpper** – conversion to upper case. A single-operand function, takes a textual type argument.

- **toLower** – conversion to lower case. A single-operand function, takes a textual type argument.

toLower and toUpper functions – useful in case of non-uniform data in the data source.

For example:

if the column $currency has values 'pln' and 'PLN', a following notation can be used for summing up these values:

sum($balance, toUpper($currency) = PLN)

- **trim** – deletes empty characters from the beginning and the end of the text. A single-operand function, takes a textual type argument.

- **reverse** – reverts the text. A single-operand function, takes a textual type argument.

- **substring** – cuts the text. Function has three or two arguments. The first argument is a text which is a basis for trimming. Remaining two arguments are integers specifying, respectively, the beginning and the end index of the new text which is the result. The length of the obtained text is the 'end index' minus 'beginning index'. In case the third argument is not provided, the function acts as if the length of the text was provided.

- **indexOf** – searches for a subtext in a given text. Two first arguments are a basic text and searched subtext respectively. The third argument is optional and denotes a place to begin the search. In case it is absent, the number 0 is used. The returned value is an integer and denotes an index of the character from which the found subtext beings or -1 in case the subtext is not found.

- **toNumber** – turns the specified text into a number (supported decimal separators are ',' and '.'.) A single-operand function.

- **dayDiff** – function with 3 arguments, the third argument is optional. The function returns a difference between the dates created on the basis of 2 first arguments. A default date format is yyyy-MM-dd. Additionally, the user can, with the use of the third argument, define the date format.

- **lastIndexOf** – performs a search for a subtext in the given text. First two arguments are base text and searched subtext respectively. The search is performed from the end. The third argument is optional and denotes a place to begin the search. In case it is absent, the number 0 is used. The returned value is an integer and denotes an index of the character from which the found subtext beings or -1 in case the subtext is not found.

- **length** – text length. A single-operand function, takes a textual type argument and returns an integer.

- **equalsIgnoreCase** – equality of texts with ignored letter case. Returns a logical value compliant with the result of comparison of two texts,

- **startsWith** – verifies whether the text is a prefix of another one. Function has two arguments. Returns a logical value informing whether the second argument is a beginning of the first one.

- **getDay** – returns day from the date. A single-operand function.

- **getMonth** – returns month from date. A single-operand function.

- **getYear** – returns year from the date. A single-operand function.

- **addDay** – adds days to date. Function has two arguments – first is a date which is a basis for modification and the second one is an integer specifying number of days. Negative number of days denotes date reduction. Function returns date increased or reduced by a specified number of days.

- **addMonth** – adds months to the date. Function has two arguments – first is a date which is a basis for modification and the second one is an integer specifying number of months. Negative number of months denotes date reduction. Function returns date increased or reduced by a specified number of months.

- **addYear** – adds years to date. Function has two arguments – first is a date which is a basis for modification and the second one is an integer specifying number of years. Negative number of years denotes date reduction. Function returns date increased or reduced by a specified number of years.

- **date** – returns a date. Function takes three arguments with integers denoting day, month and year, respectively.

- **diffDay** – interval between dates in days. Function has two arguments – takes two dates and returns an interval counted in days. Returned value is a nonnegative integer if the first date is less than or equal to the second, or negative otherwise.

- **isLeapYear** – a leap year. Checks whether the year is a leap year for a given date or an integer representing the year. Returns a logical value.

- **case** – selection function. Function has 3 arguments, the first argument is of logical type, remaining two are unspecified they must however be of a same type and such type takes a returned value. If the first argument is true then the function takes a value of the second argument, in the opposite case it take a value of the third argument.

The function is helpful in case the source for drawing data needs to be selected, depending on certain calculations.
For example:
in case there is another column included in the balance sum, depending on the record type:

sum(case($type = A, $balance1, $balance2), $type = [A, B])

then the expression sums up the records adding the value $balance1 for type A and $balance2 for type B

- **concat** – text concatenation. Function takes two or more texts which will be joined in one.

6.2.6.5 Contextual function

Contextual function return values connected with the context of actual processing (in the time of data fetching).

- **getReportDate** – returns a reporting date for report feeding (last date of the reporting month). Function has no arguments.

- **getSystemDate** – returns a current date. Function has no arguments.

- **getCVNumber** – access to a catalog variable of a name provided (only a literal is permitted) with a numerical type.

- **getCVString** – access to a catalog variable of a name provided (only a literal is permitted) with a textual type.

- **getCVDate** – access to a catalog variable of a name provided (only a literal is permitted) with a date type.

- **getCVBool** – access to a catalog variable of a name provided (only a literal is permitted) with a boolean type.

6.2.6.6 Aggregate function

Aggregate functions calculate values for the table.

First argument denotes an expression which is a basis of the function (calculated for each row), **second argument** is a logical condition (discriminator) which, when met, specifies a set of rows which are the basis for aggregate functions.

Aggregate functions (apart from the **count** function) return the value of the same type as the first argument. The first argument is mandatory.

- **sum** – sum of tabular expressions for all rows specified by the discriminator. If none of the rows meets the condition, 0 is returned. The first argument has a numerical type.

- **avg** – an average of tabular expressions for all rows specified by the discriminator. If none of the rows meets the condition, 0 is returned. The first argument has a numerical type.

- **same** – a condition of one value, returns a tabular expression value if the discriminator specified rows of the same expression values. In other case an error is returned. If none of the rows meets the condition, an error is also returned. The first argument, specifying the value returned, can be of any type.

- **one** – the condition of one row, returns a tabular expression value if the discriminator specified one row. In other case an error is returned. The first argument, specifying the value returned, can be of any type.

- **any** – a value of any row meeting the rule of discriminator. If none of the rows meets the condition, an error is returned. The first argument, specifying the value returned, can be of any type.

- **first** – a value of the first row meeting the rule of discriminator. If none of the rows meets the condition, an error is returned. The first argument, specifying the value returned, can be of any type.

- **last** – a value of the last row meeting the rule of discriminator. If none of the rows meets the condition, an error is returned. The first argument, specifying the value returned, can be of any type.

- **min** – calculated a minimum for rows specified by the discriminator. Types accepted: numerical and date. If none of the rows meets the condition, an error is returned.

- **max** – calculated a maximum for rows specified by the discriminator. Types accepted: numerical and date. If none of the rows meets the condition, an error is returned.

- **count** – calculates the number of rows meeting the condition specified by the discriminator. The first argument can be a '*' character which means that all rows meeting the condition will be calculated, or an expression of any type – then occurrences of different values of such expression will be calculated.

For example:
*count(*, $balance_owed >0)* – calculates the records with a balance greater than zero

count($account_type) – calculates a number of account types occurring in the set

Same function can be used implicitly if columns not connected with any other aggregate function are enclosed in the expression.

For this reason expressions:

$a+$b and ***same($a+$b, true)***, where *a* and *b* are column names,

are equivalent.

Some aggregate functions (min, max, sum and avg) are a basis for creating double aggregate functions.

These functions meet the criteria of aggregate functions for numerical values expanding the list of arguments by any number of grouping expressions.

Each grouping expression can be of any type and the vector specified by compounding the values of these expressions defines the subset of rows which is the basis of an internal aggregate function.

The value of an external aggregate function (which is simultaneously a value specified by the function) is calculated against the value for each vector of grouping arguments.

Name of the double aggregate function is created by composition of two aggregate functions separated by '_' (e.g. 'sum_min').

6.2.6.7 Type calculation

Language generated by the above grammar is subject to compliance verification of types (set of correct texts is limited by a required compliance with types of operations performed). Type calculation is based on the following rules:

- all literals (numerical, logical and textual) have a defined type,
- arithmetic operations require numerical operators, in case the argument type is not specified an implicit conversion is performed,
- logical operations require logical operators, in case the argument type is not specified an implicit conversion is performed,
- all comparison operations require compliant argument types; if one type is unspecified, an implicit conversion is performed; if both types are unspecified, a default type is chosen,

- comparison operations ‘**=**’ and ‘**!=**’ can be applied to all argument types; test type is the default type,
- comparison operations ‘**<**’, ‘**<=**’, ‘**>**’ and ‘**>=**’ can have arguments of numerical and date types; date type is the default type,
- types of function arguments are specified by their permitted signatures; selected signatures are defined by the type of the returned value;
- in case the function has more possible type for a given number of arguments, a version compliant with argument types is selected, in case the function has unspecified types of conclusive arguments, it remains unspecified;
- unspecified types are columns or functions which, because of unspecified parameter types, return values dependent on the required function type;
- conversion of unspecified types is performed with the use of functions (**toNumber**, **toBool**, **toDate** with default formats) for columns and conversion of ambiguous arguments for the function,
- in case propagation of rules specifying types does not end up with a defined column type – a string type is used,
- set elements have to be of the same type (automatic column conversion takes place similarly to a simple comparison operation), type of the argument for comparison with the set has to be compliant with set elements;
- access functions to catalog variables return variable type value.

6.2.6.8 Grouping functions

Grouping functions do not calculate values but define row groups which serve as the basis to set values of other expressions in the group. There are two grouping functions:

- **group** – grouping, accepts a non-empty list of grouping expressions of any type. Each argument value vector sets out a group,
- **conditionGroup** – conditional grouping, as a first argument a condition is accepted which is met by all groups specified by the function. Subsequent arguments are a list of grouping expressions of any type. Each argument value vector sets out a group.

6.2.6.9 Automatic formatting

Expression editing and validation is connected with automatic formatting which, apart from expression text alignment, can change the following elements:

- delete surplus brackets,
- remove quotation marks from textual literals if they begin with a letter, do not contain white characters and do not belong to reserved words (**true**, **false**, **not**, **if**, **then**, **else**, **let**, **in**),
- remove multiple nesting of ‘**-**’ before numerical literals,

- remove the second argument of the aggregate functions if the function is a literal and has a **true** value,
- remove a formal entry of **same** function if it does not contain additional conditions and is not nested in any other aggregate function,
- remove surplus occurrences of column conversion (in case in which an expression type is unambiguous and there is no default format provided),
- change the comparison argument so that there is a column on the left side,
- change the comparison argument so that there is a set operator on the right side.

6.2.6.10 Regular expressions

Regular expressions are used with column comparisons with a textual value.

If the text contains '**?**', '*****', '**[**' or '**]**' which are preceded by '\' these characters are treated as **special characters**. They are used for a less precise value specification of the searched column.

? replaces any (exactly one) character,

***** replaces any number (also none) of characters.

[ ] between these brackets characters from which one text character will be matched are listed

For example:

*$surname = 'J*son'* denotes a surname beginning with a letter 'J' and ending with 'son', and

$account = '11-[123]' will match the following account numbers: '11-1', '11-2' and '11-3'.

6.2.6.11 Calculating tabular expressions

To calculate a tabular expression it is necessary to load the data source content and verify the column correctness.

If an expression requires other than contextual column type, appropriate conversion is performed. It is possible then that one column will be subject to conversion to different types within one expression, depending on the context of its occurrence.

Columns (represented in the expression by names preceded with '**$**') are always linked with the most nested aggregate function and they serve as its arguments:

sum(one($a) +$b) means that column *a* is an argument for function **one**, and ***b*** – for the function **sum**.

In case of nested aggregate functions, their values are calculated only once.

Each nesting level requires an independent level of source table processing.

Calculation of expressions enclosed in the group is performed in two stages.

In the first step groups are specified on the basis of a grouping expression, and then within each group independently its dependent expressions are calculated.

Values of logical conditions (mainly discriminators) are calculated as quickly, as quickly the result is known. In order to obtain greater efficiency it is recommended to use more selective conditions in the beginning of conjunction and less selective in the beginning of alternative.

6.2.6.11 Virtual columns

Tabular data source can define virtual columns.

Virtual column value:

- is expressed by expressions of a tabular data source not containing aggregates and access to variable catalogs,
- is calculated for each row with the use of other (non-virtual) columns with values for this row,
- is added to data source and used in the same way as the ordinary column in mapping expressions. A symbol of the virtual column cannot coincide with data source columns.

6.2.6.12 Tracking of tabular data source processing

Tabular data source processing is a sequential transition through all data source rows.

It is possible to track the use of individual rows on the level of a given expression which means that all rows used in valuation of a given expression can be selected from the source.

A row is used by a mapping expression if it meets a condition occurring in at least one grouping function of the first level (non-nested).

This functionality is used in the application in two cases:

- mappings verification – possibility to view all rows which took part in calculation of a selected cell – individually for each row of the data source used,
- tracking of unused records – possibility to record all rows which were not used by any mapping expression from the currently calculated expression set – individually for each row of the data source used.

6.2.6.13 Data sources of Excel cell type

Data source enables fetching Excel file(s) of any structure. Data source construction consists of creating a table in which each row represents a specified cell from the file. Data source uses the same mechanisms as other tabular data sources. An additional mechanism enables direct addressing of Excel file cells with the abbreviation:

@file_name!spreadsheet_name!cell_name

where file name and spreadsheet name are optional. However, the expression assumes that the maximum of one cell with a provided address is present in the source. Addressing expression:

@file.xls!sheet1!A1

is equal to:

oneOrNull($sys.value, $sys.value != '' && $sys.address = 'A1' && $sys.sheet_name = 'sheet1' && $sys.file_name = 'file.xls')

6.2.7 Importing and exporting of mapping expressions

It is possible to export all expressions from the selected table subset for a given mapping set.

Such expressions are addressed with individual facts so they can be imported to another taxonomy if the facts specified are included in it.

Data sources are co-shared between the mapping sets that is why they are not exported and their presence is simultaneously required for the newly imported expressions.

An exported file format is compatible with the one accepted at the time of import. The format is described below with XMLSchema:

```
<xs:schemaid="fact-mappings"xmlns=""xmlns:xs="http://www.w3.org/2001/XMLSchema">
<xs:elementname="fact-mappings">
<xs:complexType>
<xs:sequence>
```

```
<xs:elementname="fact-mapping"minOccurs="0"maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:elementname="fact"type="xs:string"minOccurs="1"
maxOccurs="1"/>
<xs:elementname="expression"type="xs:string"
minOccurs="1"maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
<xs:elementname="tuple-mapping"minOccurs="0"maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:elementname="tuple-item"minOccurs="0"maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:elementname="group-expression"type="xs:string"
minOccurs="0"maxOccurs="1"/>
<xs:elementname="fact-mapping"minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:elementname="fact"type="xs:string"
minOccurs="1"maxOccurs="1"/>
<xs:elementname="expression"type="xs:string"
minOccurs="1"maxOccurs="1"/>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attributename="id"type="xs:string"use="required"/>
</xs:complexType>
</xs:element>
<xs:elementname="mapping-context"minOccurs="0"
maxOccurs="1">
<xs:complexType>
<xs:sequence>
<xs:elementname="typed-dimension-mapping"minOccurs="0"
maxOccurs="unbounded">
<xs:complexType>
<xs:sequence>
<xs:elementname="member"minOccurs="0"maxOccurs="unbounded">
```

```
<xs:complexType>
<xs:sequence>
<xs:elementname="member-expression"type="xs:string"
minOccurs="1"maxOccurs="1"/>
</xs:sequence>
<xs:attributename="id"type="xs:string"use="required"/>
</xs:complexType>
</xs:element>
</xs:sequence>
<xs:attributename="id"type="xs:string"/>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:sequence>
</xs:complexType>
</xs:element>
</xs:schema>
```

6.2.7.1 Description of elements included in the XML Schema

</xs:schema>

Element <fact-mapping>

- the main level element containing mappings, has got an optional attribute describing the mappings version.

Element <ds-context>

- the element which allows to declare the sources used in a given set of mappings. All sources used in the set should be listed in the next parts of this type. The parent of this element is **<facts-mapping>.**

Element<fact-mapping>

- definition of the element describes the mapping of a single cell.

The parent of this element is <facts-mapping>.

The composition of this component consists two subelements:

- element <fact>, allowing for indicating fact from XBRL instance.

The fact address in the XBRL instance is built according to the following BNF notation:

<taxonomy_code>;<measure code>;<dimensions code>;;<tag period>;<value>

where:

<taxonomy_code> The unique identifier of the taxonomy type. This identifier is visible when the set in report is highlighted. Examples of identifiers: C - COREP CRD III, LQITS.

<measure code> is a clear definition of the value measure.

The role of measures codes and dimensions play Polish technical labels or for taxonomy without technical labels- XML ID id from the element defining the concept.

<dimensions code> is a unique term for the combination of dimensions and their components. In the case of reported value in the context of multiple dimensions, individual codes are separated by a comma:

<dimension code>,<dimension code>,<dimension code>…

<dimension code>is built from the identifier or Polish technical label dimension and from the tag of component dimension, separated by a colon:

<ID/label dimension>:<tag of component dimension>

<tag of component dimension> for predefined dimensions it's an XBRL identifier or technical label component.

For the dimensions defined by the user (typed dimensions) identifier is built from the current component value of the dimension, according to the following algorithm:

- for components made up of several fields - their content is combined into a single record, separated by an underscore,

  - In the resulting record whitespace and characters colon, comma, colon, backslash, minority, majority, quotation mark and apostrophe are converted to underscore.

<dimension code> may be null value

  - Element **<expression>** is specifying the mapping value for the given fact.

Mapping language has been described in previous chapters.

Element <shared-fact-mapping>:

- element which allows to save an identical mapping for multiple cells.
- it works like **<fact-mapping>,** however, it allows for identification many facts, for which it is defined the expression.

Element <tuple-mapping>:

- element which is grouping mapping for the facts from "tuples", which includes two sub-elements:
  - **<grouping-expression>**defines the value of the grouping function for tuples (called. tuple),
  - **<fact-mapping>** expression which defines mappings for particular member of the tuple.

The way to describe this element is identical just like for element: <fact-mapping> which is a child of **the <fact-mappings>**.

Element <mapping-context>:

- an element which allows for defining mappings of typed dimensions,

Similarly to <tuple-mapping> defined is the value of grouping function.

For typed dimensions, whose members comprise more than one component, necessary is to provide a mapping for each of them.

6.3 Snapshot storage

The following terms are connected with the snapshot storage:

Snapshots built on the basis of table indication and data sources for these tables.

The user has a possibility to use the snapshots created in the ETL tool which is built into the aSISt application.

The definition of subsequent ETL processing tasks is stored in the aSISt database.

Snapshot instance – created when the data feed process is performed.

Snapshot instance data are stored in a separate DerbyDB database.

Due to potentially great amount of such data there is an independent database created for each period. Tables created in the database are prefixed with a definition version and data version.

Storing of historical data allows for a "mapping verification" process for historical reports and definitions. This process is used for verification of values included in the report against the source data calculation results.

6.4 ETL

ETL (Extract, Transform, Load) is a tool enabling transformation of source data before their use in the mapping module.

Transformation consists of steps defined by the users in the ETL editing window.

Steps download data from table defined in the snapshot. Each step (excluding input) uses data from one or more input steps.

Most steps have columns and indexes describing data processing result.

6.4.1 Output table

A step corresponding to the table defined in the snapshot. This step is read-only. After snapshot definition change the steps need to be refreshed by opening an ETL window and, accordingly to the need, changing definitions of steps depending on the modified input step.

6.4.2 SQL instruction

A step performed for the SQL query provided by the user. The result is created by adding a prefix with **insert** instruction to the query.

6.4.3 Row script

Groovy script performed for each data row from the preceding step. At the script entry the user receives a map with the content of the processed row. As the result, a map is created, which complies with the defined column structure in the current step.

6.4.4 Table script

Groovy script is performed once for the entire table. The script allows performance of SQL queries and any data manipulation. The script itself should fill in data of the current step.

6.4.5 Row validation

A step setting in motion the set of validators verifying row correctness with the use of validators:

- Non-empty table,
- Groovy script.

6.4.6 Table validation

A step setting in motion the set of validators verifying table correctness with the use of validators:

- Non-empty table,
- Groovy script,
- SQL.

6.5 Work in a multiple user environment

In data mapping module it's possible to work simultaneously on multiple aSISt workstations connected with a single database.

This version of aSISt always requires:

- **Oracle** database

On a multi workstation version it's possible to:

- work at the same time on different mappings sets and sources and there are no restrictions in their editing.

It is not possible to simultaneously create sets of mappings and sources and editing them by multiple users.