

User Guide



Contents

Graph Workflow Architecture.....	4
ELT Maestro Client	5
Starting Client.....	5
The Workspace	6
Workflow (Job).....	8
Creating New Workflow/Job.....	8
Editing Workflow	8
Workflow Designer / Editor	9
Introduction	9
Workflow Operations.....	10
Adding/Deleting/Editing Steps & Flow Control	10
Editing Mapping / Flow	12
Running Workflow & Analyzing Log.....	13
Analyzing Historical Logs.....	14
Steps / Nodes	15
SQL Script	15
SSH	16
Sync	18
Switch.....	19
FileWatch	Error! Bookmark not defined.
JDBCWatch	21
Set Variable	22
Watermark	25
SFTP.....	26
SFTP2S3 -WIP	27
Control Test.....	28
JobStep.....	29
File Loader [Snowflake].....	30
Onstage Group [Snowflake]	31
Table [Snowflake].....	33
File Loader [Netezza].....	34
Onstage Group [Netezza].....	35
Table [Netezza]	37
Onstage Group [Spark SQL].....	38

Data Frame [Spark SQL]	40
Local File [Spark SQL]	42
Spark Data Cache / Persistence [Spark SQL]	42
Dedupe	43
Minus	44
Union	45
Projection (not available)	48
Aggregate	49
Filter	50
Join	52
Function	58
SCD2	60
Smart Script	64
Support:	Error! Bookmark not defined.

Graph Workflow Architecture

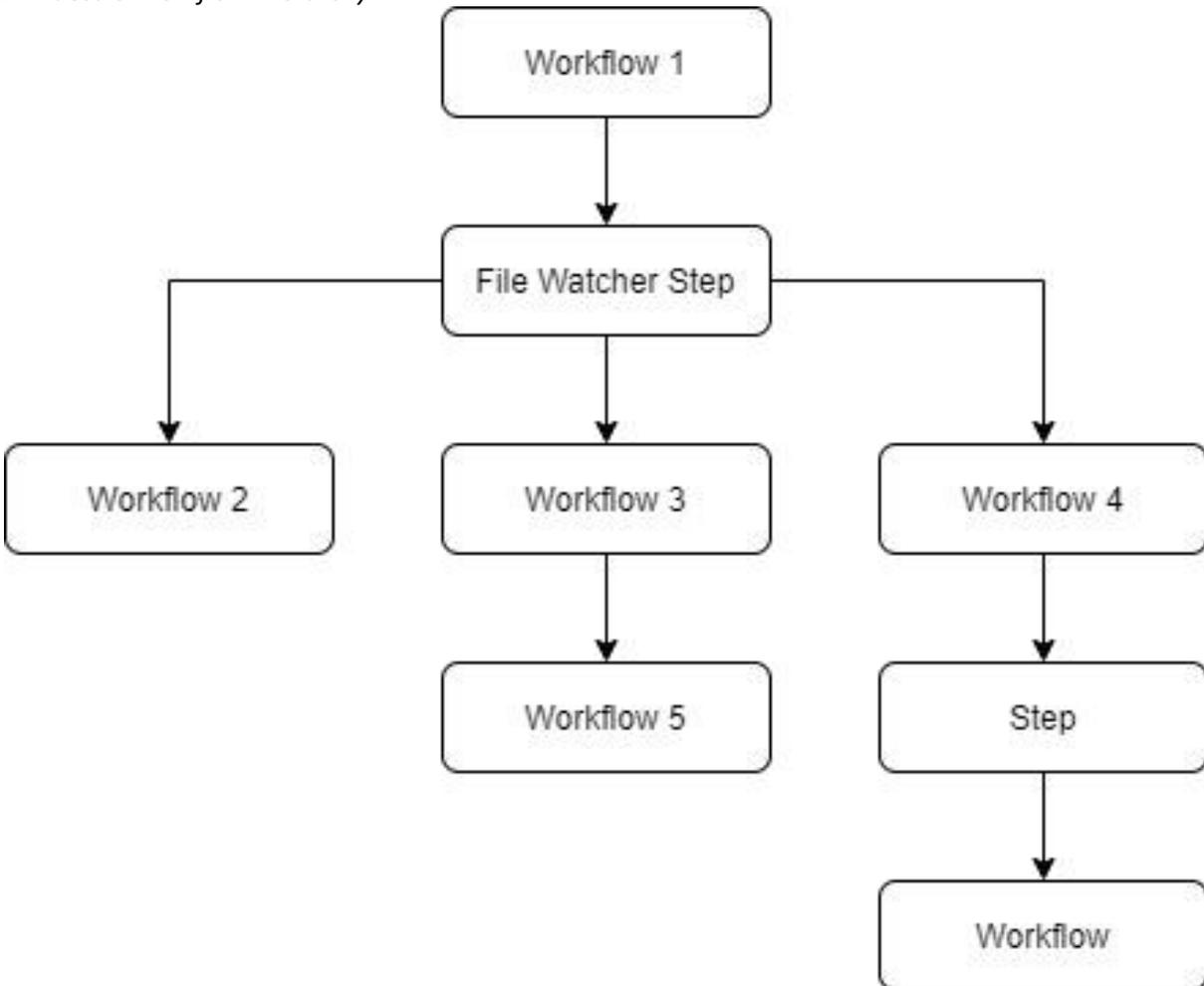
ELT Maestro workflow architecture has nodal hierarchy. For example, a job or workflow can be a node, a step can be a node, a step of type job can also be a node. There is no limitation on hierarchy level.

For example,

You can design a workflow which can call or second hierarchy level workflow which can execute third hierarchy level workflow ... and so on. There is no limitation on hierarchy level.

Consider following hypothetical workflow structure diagram.

Figure: ELT Maestro Workflow Hierarchy

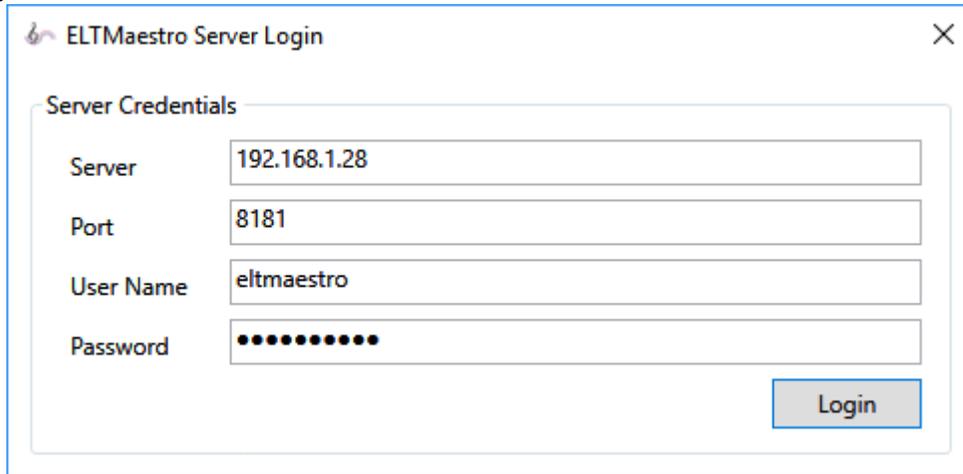


ELT Maestro Client

Starting Client

When you launch the ELT Maestro Data Integration Client, the Login screen appears as shown below:

Figure: Login Dialog



Parameters:

Property	Type	Info
Server	Text	Hostname or ip address of ELT Maestro server.
Port	Text	ELT Maestro server messaging port. Default is 8181.
User Name	Text	ELT Maestro data integration username. The credentials are not affiliated with other accounts.
Password	Text	ELT Maestro data integration user password.

- Click [Login] after entering required credentials.

Upon successful login the Workspace Window should appear. You can then start creating workflow jobs and build dataflow diagrams.

The Workspace

All created jobs loaded from workspace folder appear on this screen. Workflow jobs can be created or edited from this screen. This is the first window that appears after successful login.

Figure: Workspace Window

The screenshot shows the ELT Maestro -Workspace application window. The title bar reads "ELT Maestro -Workspace". Below the title bar is a menu bar with "File", "Administration", "Runtime & Logs", and "Help". The main content area is titled "WorkFlow(s)" and contains a toolbar with five icons: a pencil (edit), a refresh symbol, a trash can (delete), a wrench (configure), and a gear (designer). To the right of the toolbar is a search box with a magnifying glass icon. Below the toolbar is a table of workflows. The table has columns for workflow name, owner, and success rate/count. The workflows listed are:

Workflow Name	Owner	Success Rate	Success Count
JDBC_TO_NETEZZA_LOAD	INTEGRATOR	100%	7
JOB_CHILD1	INTEGRATOR	100%	8
JOB_CHILD2	INTEGRATOR	100%	8
JOB_GRANDPARENT	INTEGRATOR	88%	7
JOB_PARENT1	INTEGRATOR	100%	8
JOB_PARENT2	INTEGRATOR	88%	7
RPT_NZ_BATCH_STATISTICS	INTEGRATOR	100%	1
TEST_RK_IMPORT_NTZ	INTEGRATOR	100%	2
DELTA_IMPORT_JDBC2HDFS	INTEGRATOR	100%	1
IMPORT_TABLES_TO_HDFS	INTEGRATOR	100%	3
REFRESH_IMPORT_JDBC2HDFS	INTEGRATOR	100%	2
RPT_EMPLOYEE_COUNT_BY_CITY_SPARK	INTEGRATOR	98%	Success Count

Below the table is a section titled "ELT Maestro Server Connection" with input fields for "Server" (127.0.0.1), "Port" (8181), and "User" (integrator).

Quick launch buttons



From Left to Right

1. Create workflow
2. Refresh workflow list
3. Delete selected workflow(s)
4. Edit workflow configuration
5. Open selected workflow designer

Workspace Menu Items

File -> Create Workflow	Creates workflow
File -> Exit	Exits application
Administration -> Configure	Resource configuration
Administration -> Metrics Configuration -> Control Test	Configure control test metrics
Administration -> Scheduler	Cron scheduler interface
Administration -> Migration -> Export	Exports workflow to local system
Administration -> Migration -> Import	Imports workflow from local system
Runtime & Logs -> Logging -> Workflow Logs	View workflow log history
Runtime & Logs -> Logging -> Server Logs	View ELT Maestro server logs
Runtime & Logs -> Reports -> Metrics Report	View metrics report
Runtime & Logs -> Runtime Operations	Manage ELT Maestro process(s)
Help -> About	View product information

Daily Warehouse Metrics

This tab provides system wide summary of historical metrics. Data is aggregated by day and calculations performed on file size and row counts processed using ELT Maestro.

Figure: Daily Warehouse Metrics Report

The screenshot shows the 'Daily Warehouse Metrics' report in the ELT Maestro -Workspace application. The report is divided into two sections, each with a 'Refresh' button.

Section 1: Daily Warehousing Metrics

DATE	TOTAL_WORKFLOWS	ROWS_DOWNLOADED	NETWORK_DATA_GB	ROWS_DELETED	ROWS_UPDATED	ROWS_INSERTED
2020-04-20	1	0	0.00	0	0	0
2020-04-19	1	0	0.00	0	0	0
2020-04-18	2	39	0.00	0	0	0
2020-04-17	1	0	0.00	0	0	0
2020-04-16	4	695	0.00	0	0	205
2020-04-15	1	0	0.00	0	0	0
2020-04-14	6	2058452	0.18	114	0	223
2020-04-13	2	2692536	0.11	0	0	0

Section 2: Daily Warehousing Metrics (by feature type)

DATE	STEP_TYPE	TOTAL_WORKFLOWS	ROWS_DOWNLOADED	NETWORK_DATA_GB	ROWS_DELETED	ROWS_UPDATED	ROWS_INSERTED
2020-04-15	DATAFRAME	1	0	0.00	0	0	0
2020-04-15	JOIN	1	0	0.00	0	0	0
2020-04-15	SPARKDATACACHE	1	0	0.00	0	0	0
2020-04-14	AGGREGATE	1	0	0.00	0	0	0
2020-04-14	DATAFRAME	1	0	0.00	0	0	0
2020-04-14	JOIN	1	0	0.00	0	0	0
2020-04-14	ONSTAGEGROUP	4	2058452	0.18	114	0	212
2020-04-14	SPARKDATACACHE	1	0	0.00	0	0	0
2020-04-14	TABLE	1	0	0.00	0	0	11
2020-04-13	ONSTAGEGROUP	2	2692536	0.11	0	0	0

Workflow (Job)

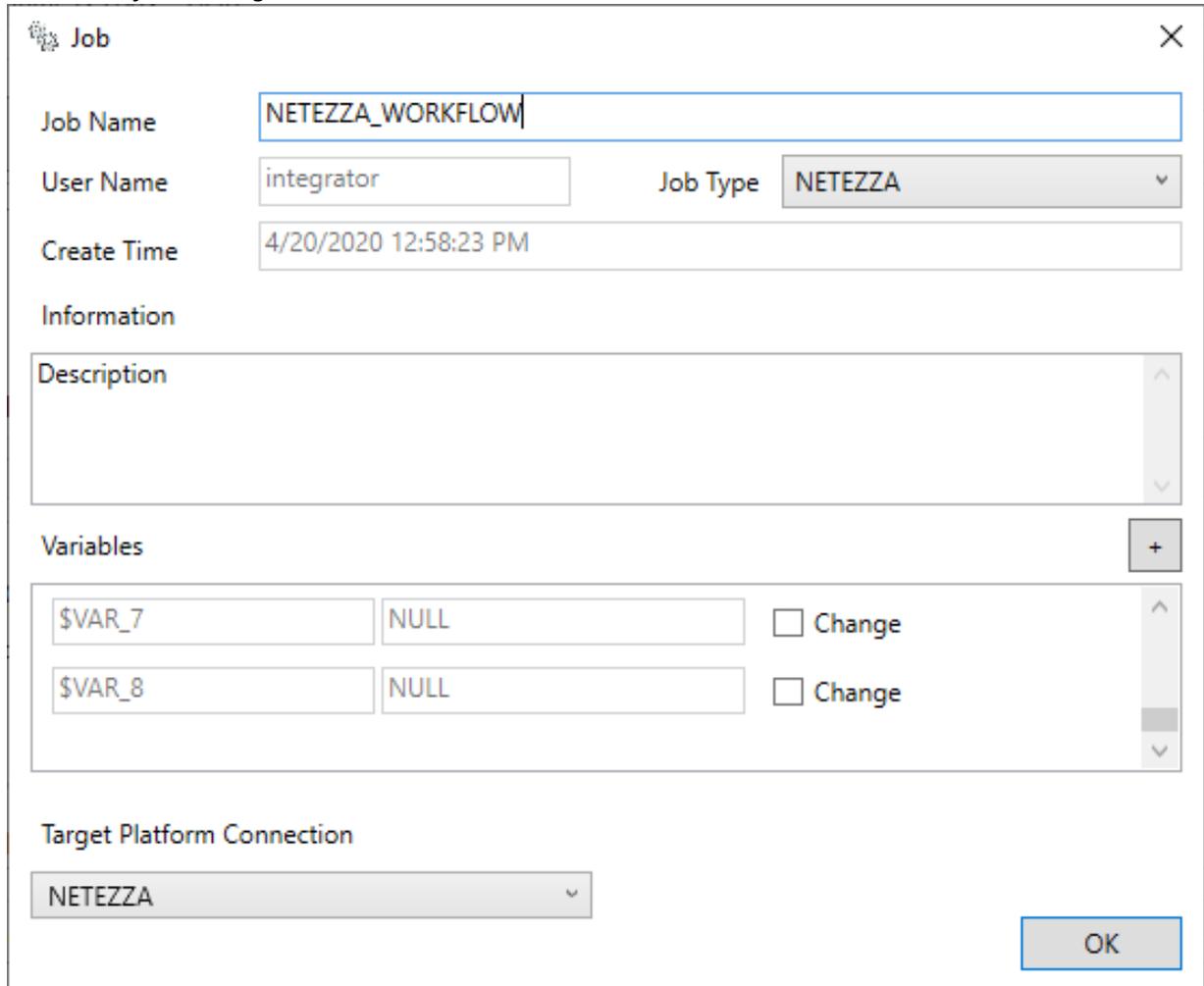
Creating New Workflow/Job

Most of ELT mapping design happens on a workflow. Jobs contain steps and mapping lines. Steps are the smallest purpose-driven objects. Mapping lines link the steps together, control parallelism, order data flow and map columns.

- Click [Create]  button on workspace to open a new job dialog.
- Add [Job Name], select [Job Type], select [Target Platform Connection] then Click [OK]

Once workflow is created a message box may appear asking if you want to edit/open workflow. Click [Yes] to continue working on created workflow.

Figure: Create Workflow Dialog



Job [X]

Job Name: NETEZZA_WORKFLOW

User Name: integrator Job Type: NETEZZA

Create Time: 4/20/2020 12:58:23 PM

Information

Description

Variables +

\$VAR_7	NULL	<input type="checkbox"/> Change
\$VAR_8	NULL	<input type="checkbox"/> Change

Target Platform Connection

NETEZZA

OK

Job Variables

Default variables are created when creating a new workflow. Variables with name \$VAR_ can be modified during runtime using "Set Variable" step.

Editing Workflow

- On main workspace window, Double Click on workflow name to open editor or
- Select workflow and then Click  to open editor.

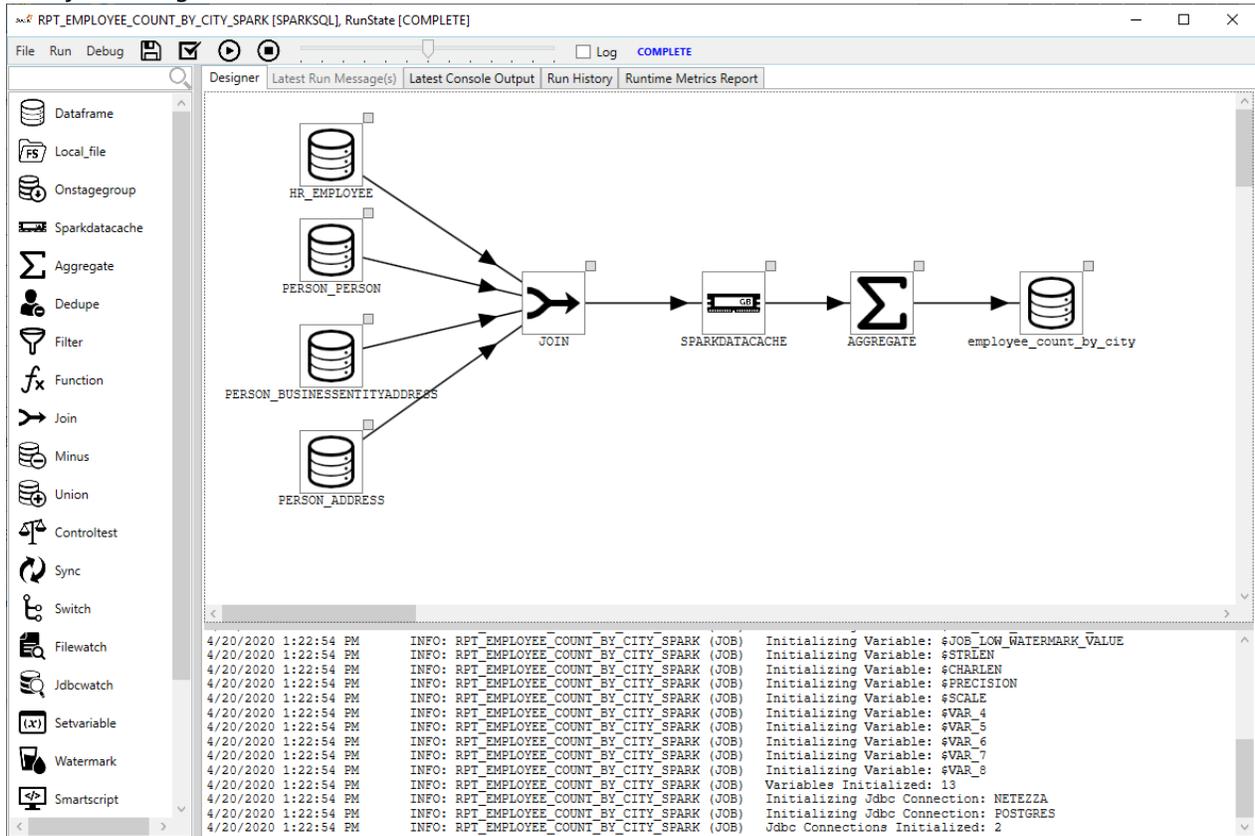
Workflow Designer / Editor

Introduction

Workflow designer or job editor contains tools required to designed ELT workflow.

1. Left panel contains list of Steps (also called nodes) which can be dragged and dropped into canvas area.
2. Right-Center panel is the canvas area that represents workflow item(s).
3. Bottom-Right panel contains workflow related messages.

Figure: Workflow Designer



Workflow Tab Items

Designer contains main workflow canvas where development work happens.

1. [Last Run Message] displays latest runtime log for selected step (when Log is checked)
2. [Last Console Output] displays last console log
3. [Run History] displays it's historical execution states
4. [Runtime Metrics Report] displays count metrics and other measurements based on recent execution

Quick Launch Buttons (from left to right)



1. Save Workflow
2. Check for Workflow Mapping Errors
3. Run Workflow
4. Stop/Terminate Workflow

Logging (recent execution state)



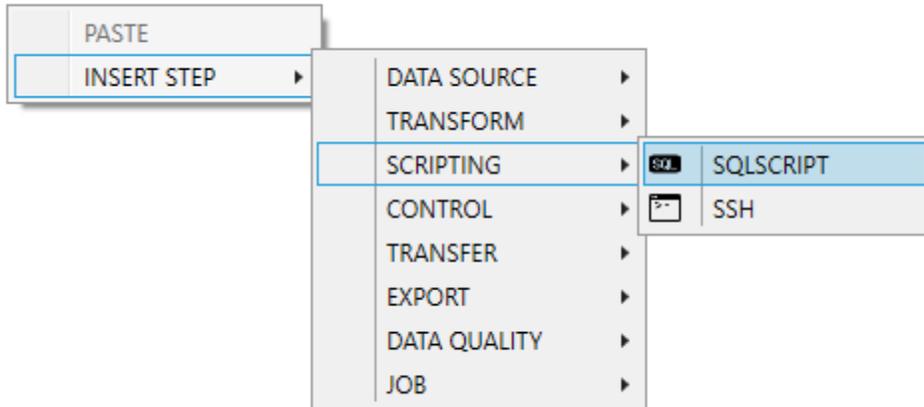
Log View mode times out after 1 minute automatically. You must uncheck then check mark Log to refresh.

Workflow Operations

Adding/Deleting/Editing Steps & Flow Control

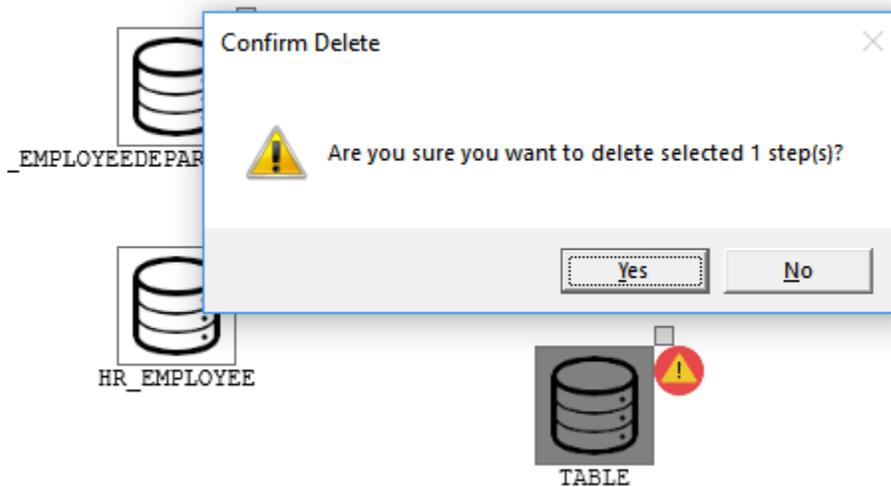
Adding a step

- [Right] click on canvas and [select] step to add on cursor position.
- or drag-drop a step from left panel



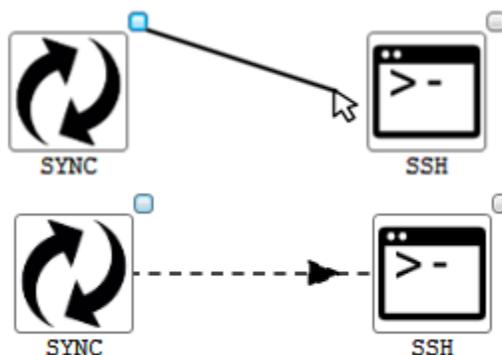
Deleting a step

- [Left] single click to select a step, then click [Delete], then confirm [Yes]/[No]



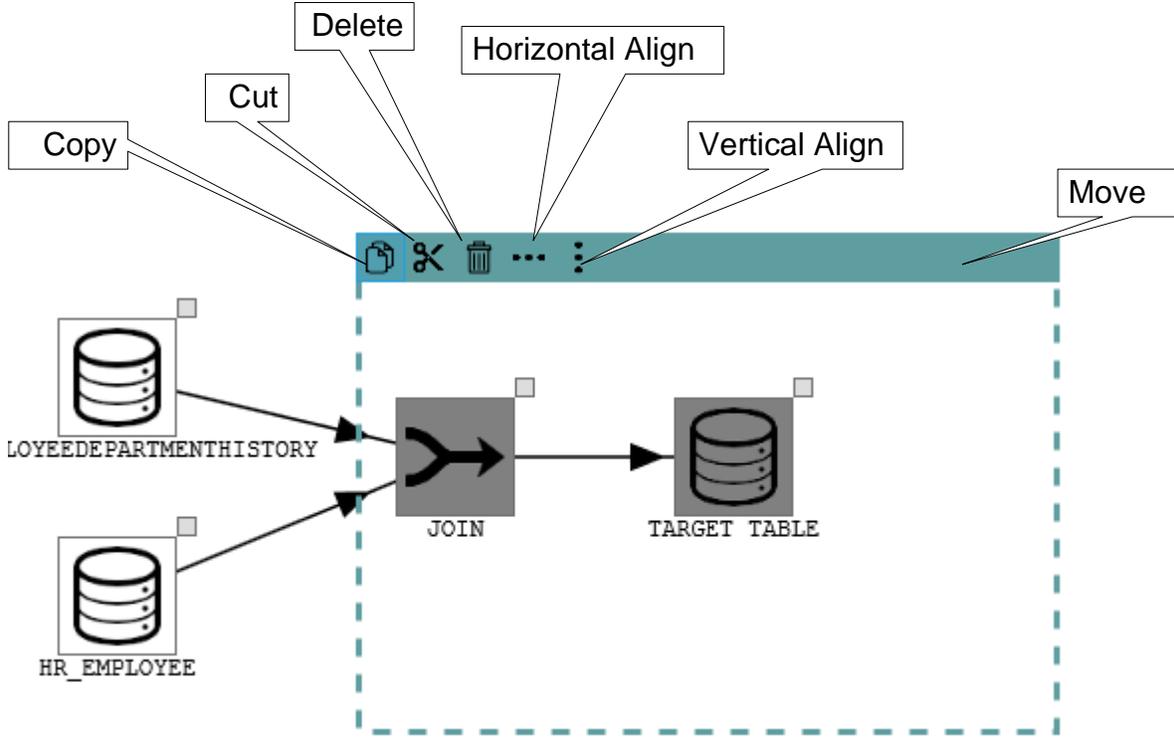
Adding Flow Line / Mapper (Connecting Steps)

- While holding mouse button move cursor on target step and release.



Group Actions

- Select multiple steps by creating a rectangle on canvas, then perform action.



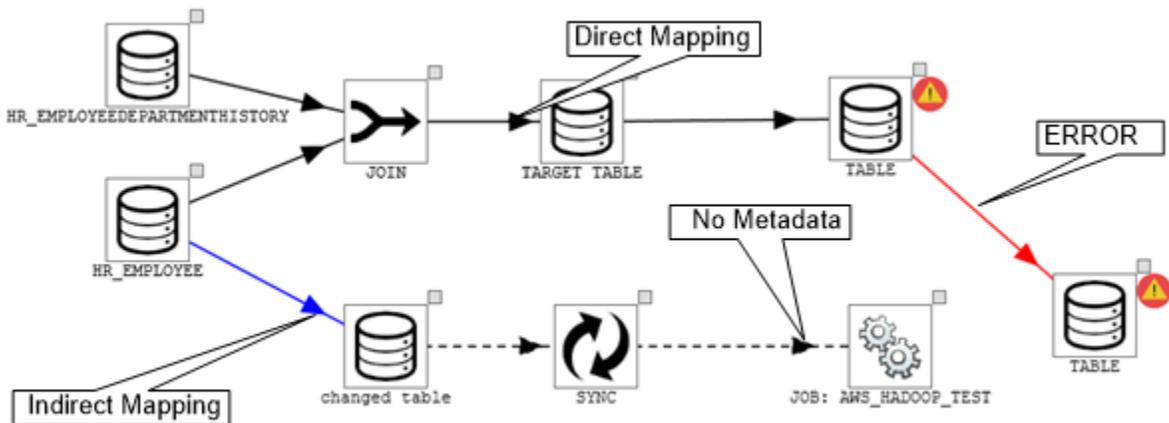
- To paste copied or cut objects, [Right] click on canvas and click [Paste]

Line Color & Metadata

1. RED No metadata found when connecting data steps
2. BLUE At least one column mapped from source step to target step has different name
3. Solid BLACK All columns are mapped with same name on source and target steps
4. Dotted BLACK Metadata is not required. This is an execution flow.

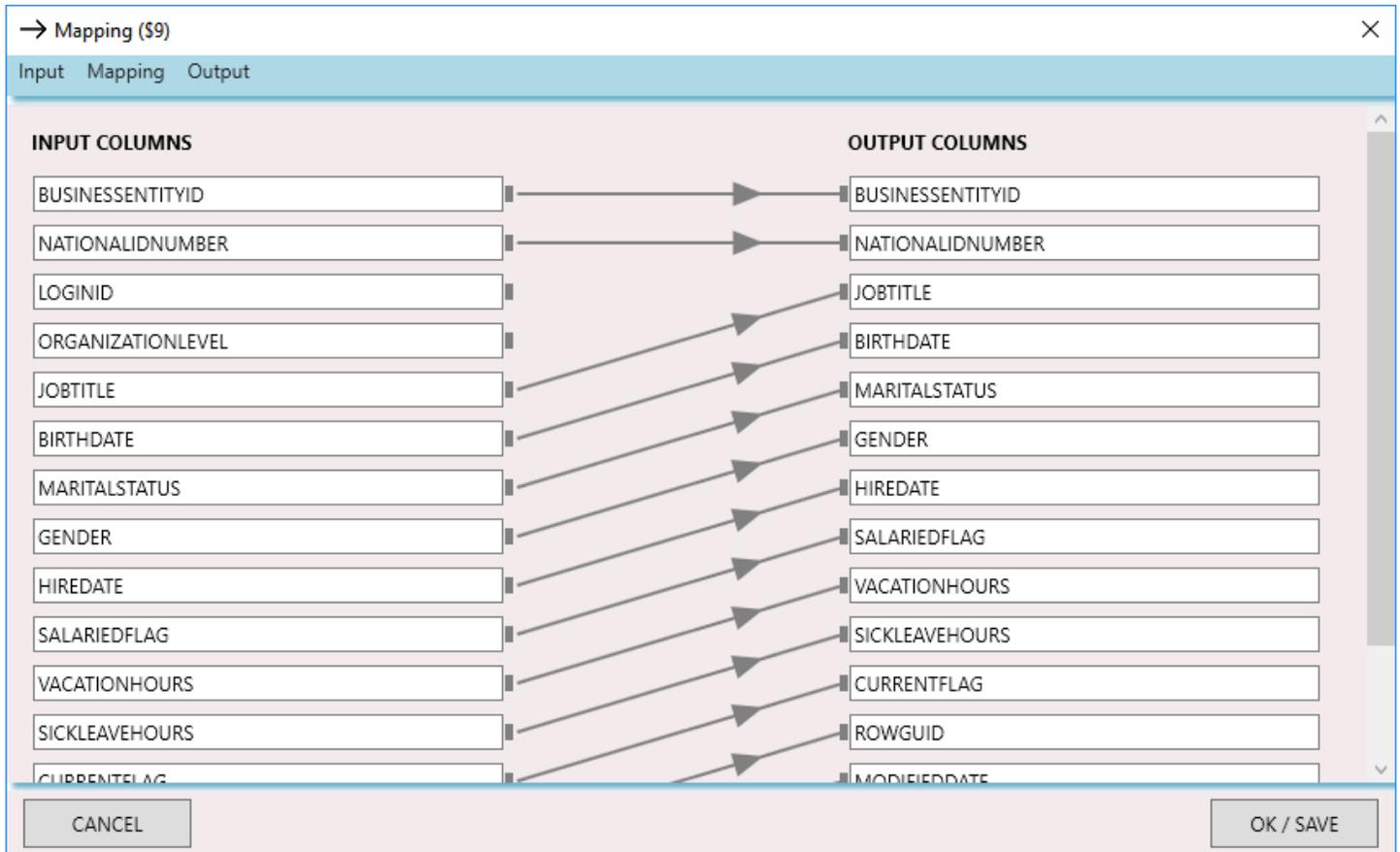
By default the arrow direction indicates control flow. In addition, the solid arrows indicate data flow.

Figure: Line Colors



Editing Mapping / Flow

- [Click] on the line arrow head to open mapping editor.



Menu Items Description

1. [Input -> Refresh] refreshes incoming column names and data types
 2. [Input -> Push to Output] refreshes output column names
 3. [Mapping -> Clear] removes column mapping linkers
 4. [Mapping -> Guess Mapping -> Option] draws linkers based on selection
- [Output -> Refresh] refreshes output columns list based on target step's metadata

Running Workflow & Analyzing Log

- Click [Play Button] to run current workflow.

Additionally you will receive a dialog for run configuration. When workflow is running Log Mode is checked by default. You may click on status icon to view detailed message about specific step.

Figure: Running Workflow

The screenshot shows the workflow designer interface for a workflow named 'RPT_EMPLOYEE_COUNT_BY_CITY_SPARK [SPARKSQL]'. The workflow is in a 'RUNNING' state. The designer shows a sequence of steps: four Dataframe inputs (HR_EMPLOYEE, PERSON_PERSON, PERSON_BUSINESSENTITYADDRESS, and PERSON_ADDRESS) feeding into a JOIN step. The output of the JOIN step goes to a SPARKDATACACHE step, followed by an AGGREGATE step, and finally an output Dataframe named 'employee_count_by_city'. The console log at the bottom shows the following output:

```
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $STRLEN
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $CHARLEN
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $PRECISION
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $SCALE
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $VAR_4
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $VAR_5
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $VAR_6
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $VAR_7
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Variable: $VAR_8
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Variables Initialized: 13
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Jdbc Connection: NETEZZA
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Initializing Jdbc Connection: POSTGRES
4/20/2020 1:22:54 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Jdbc Connections Initialized: 2
4/20/2020 2:04:40 PM INFO: RPT_EMPLOYEE_COUNT_BY_CITY_SPARK (JOB) Run requested
```

Figure: Join Step Output

The screenshot shows the workflow designer interface with the console log for the JOIN step selected. The log shows the following output:

```
2020-04-20 14:04:47 INITIALIZED
2020-04-20 14:05:13 RUNNING
2020-04-20 14:05:13 [Run Join Sql -> DataFrame] SELECT dataframe_5219_personaddress_57.'CITY' AS `CITY`, dataframe_5219_hremployee_78.'BUSIN
[Persist Results] -> StorageLevel(disk, 1 Replicas) With Rowcount Of 290 Rows
Rows: 290
2020-04-20 14:05:15 DataFrame -> View [join_5219_join_87]
2020-04-20 14:05:15 COMPLETE
```

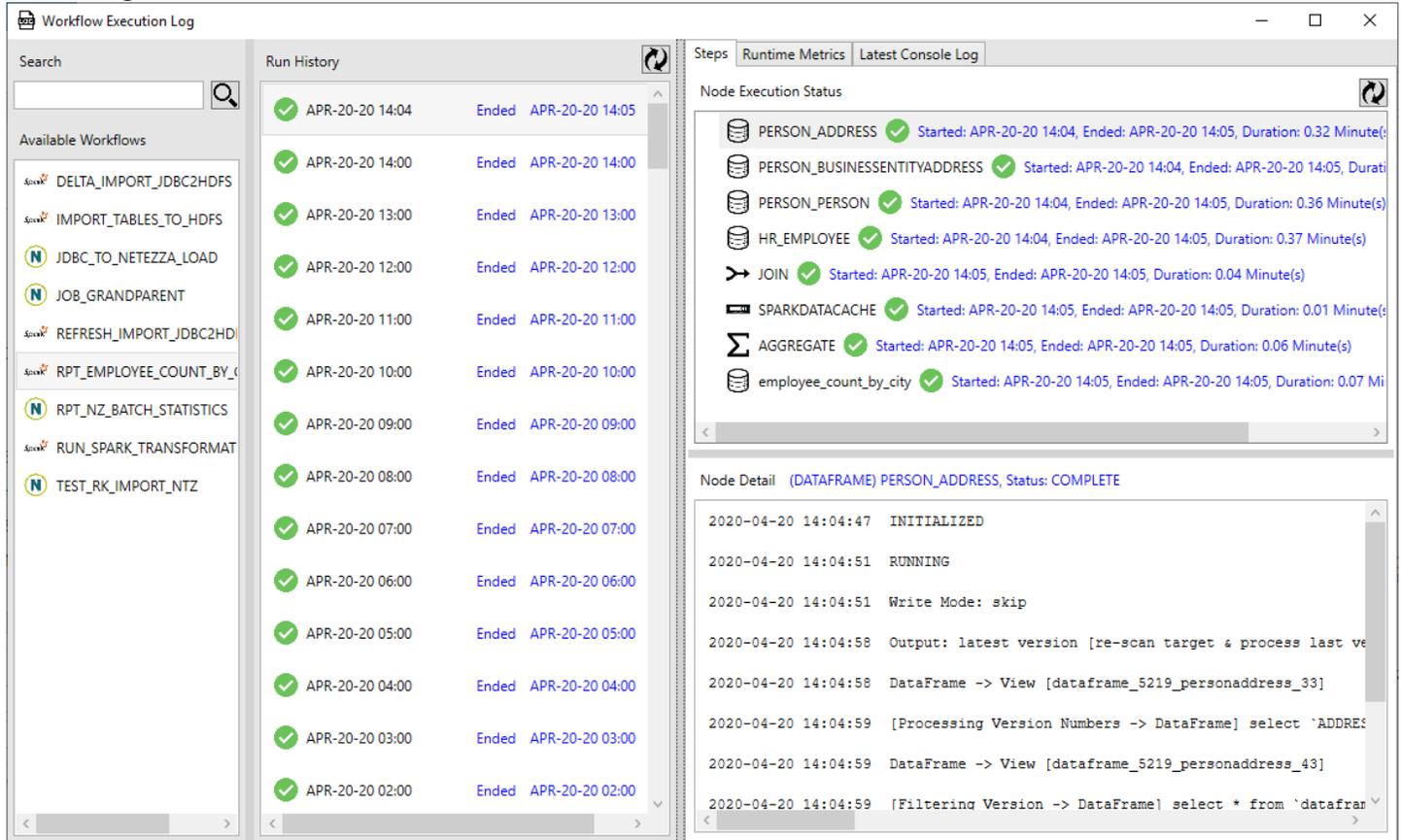
Console log is also available on [Latest Console Output] tab.

Analyzing Historical Logs

Open log viewer from Workspace/Main Window

- Click [Runtime & Logs] -> [Logging] -> [Workflow Logs]

Screen: Log Viewer



Panels

1. Left Panel contains workflow names
2. Middle Panel contains run history by date time
3. Top Right Panel contains nodes executed
4. Bottom Right Panel contains selected node messages

Other Tabs

1. Runtime Metrics contains object metrics counts, like rows, bytes etc for that run date/time
2. Latest Console Log contains last logfile content

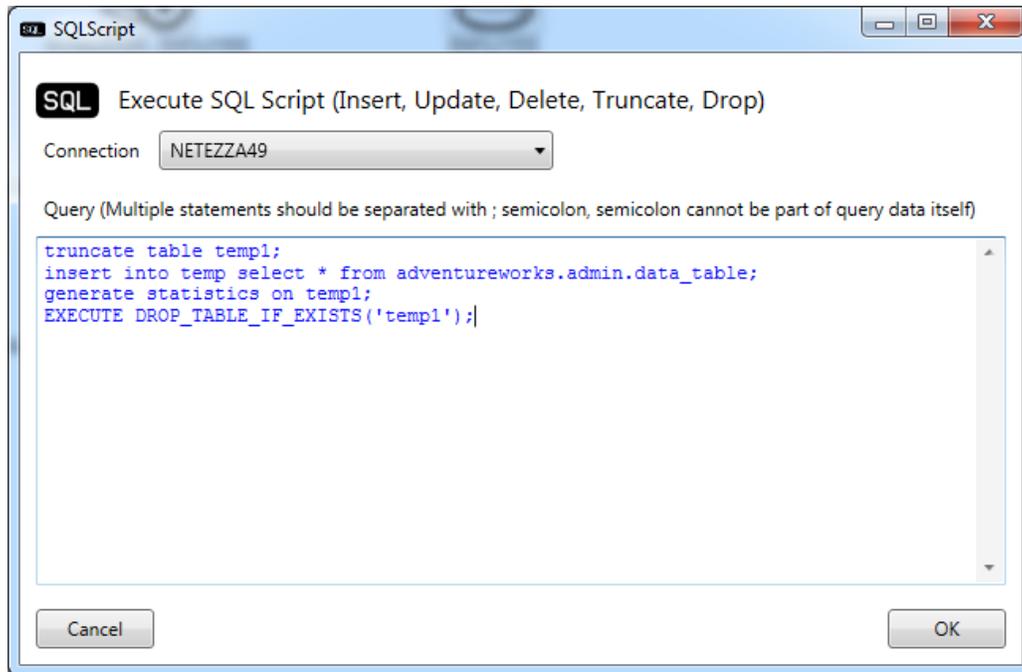
Steps / Nodes

SQL Script

Introduction

SQL Script step enables executing SQL queries on a JDBC connection. Statements are separated by semi colon.

Example,



SQL Statements

Multiple SQL statements are separated by semi-colon. Semi-colon cannot be part of query statement. For example, using following statement will result in error during runtime.

```
insert into MY_DATA_TABLE select COLUMN1, COLUMN2||'added;' from  
MY_SOURCE_DATA_TABLE
```

While above statement is still valid, runtime module splits entire content with semi-colon into multiple statements which leads to one or more invalid SQL.

Calling Stored Procedures

Executing procedures in SQLScript step is also possible which can be achieved by simply inserting call procedure statement. Stored procedure calls only work on your targeted platform connection.

Example,

```
EXECUTE DROP_TABLE_IF_EXISTS('temp1');
```

SSH

Introduction

The SSH step runs a shell command on one of the Spark nodes. You can use the result of the shell command to control the execution of a subsequent step.

Example,

We set up our job as follows:



In the properties of the SSH command, we enter the following:

The screenshot shows the 'Execute Shell Command (SSH)' dialog box. The title bar reads 'Execute Shell Command (SSH)'. The main title is 'Execute SSH Shell Command'. The 'Command' field contains the following text:

```
ps -aef
~/scripts/run-another-process.sh
```

The 'Success/Failure Basis' section has three radio buttons: 'Exit Status', 'Normal String' (which is selected), and 'Failure String'. Below this is a text field for 'Scan Following Text' containing the word 'COMPLETED'. The 'Linux Server Information' section has a 'Connection Name' field with 'HADOOP1' and a 'Browse' button. A note at the bottom of this section reads: 'ELTMaestro runtime engine first attempts to establish direct connection to specified ssh connection before using any available agents.' At the bottom of the dialog are 'Cancel' and 'OK' buttons.

where run-another-process.sh is a shell script that may or may not echo the string 'COMPLETED' to standard output. When the script indeed produces the string 'COMPLETED', the step is successful, which then causes the DEDUPE_TEST job to run; this action can be observed on the job canvas with Debug mode enabled:



If we alter the script so that it no longer echoes 'COMPLETED', the SSH step fails, and the DEDUPE_TEST job does not run.



Success or Failure Options

Exit Status

If exit status code of shell script equals zero then set this step status to SUCCESSFUL, otherwise FAIL.

Normal String

If output contains mentioned string to be scanned set step status to SUCCESSFUL, otherwise FAIL.

Exit status is ignored if this option is checked.

Failure String

If output of script contains mentioned string to be scanned set step status to FAIL, otherwise SUCCESSFUL.

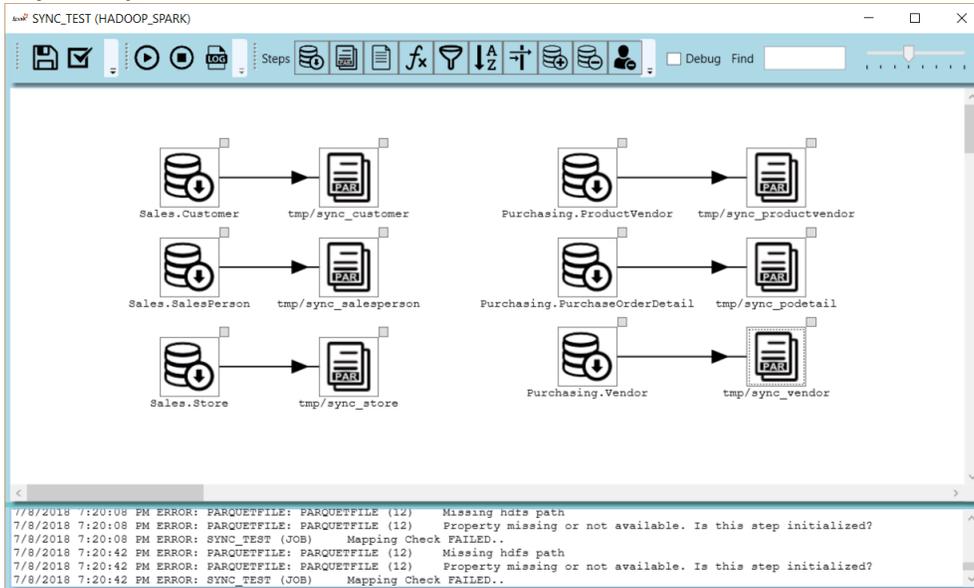
Exit status is ignored if this option is checked.

Sync

Introduction

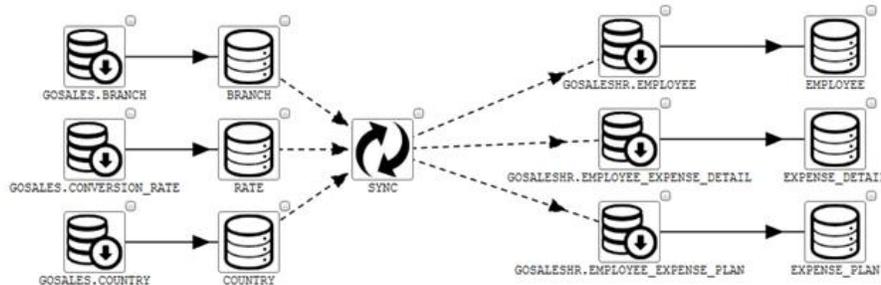
Sync is a dummy step which always returns successful state if executed. The purpose of this step is to control ELT parallelism for certain MPP systems that contain high throughput and low concurrency. Sync step does not have any configuration property.

Example, (Without Sync Step)



Above workflow executes 6 workflow paths in parallel, which means the number of connections on source and target amounts to 12 (6 on source, 6 on target). This number can easily go very high when adding more tables. To implement phasing mechanism which is to load few, wait, load more, repeat kind of operation Sync step should be considered when designing parallel processes.

Example,



Above workflow executes in following order:

1. Load three tables in parallel.
2. Sync waits for completion.
3. Load three more in parallel.

Note

ELT Maestro engine dynamically manages all connection requests to ensure workflow process is not reserving any idle connections at any given instant to free up those resources automatically.

Switch

Introduction

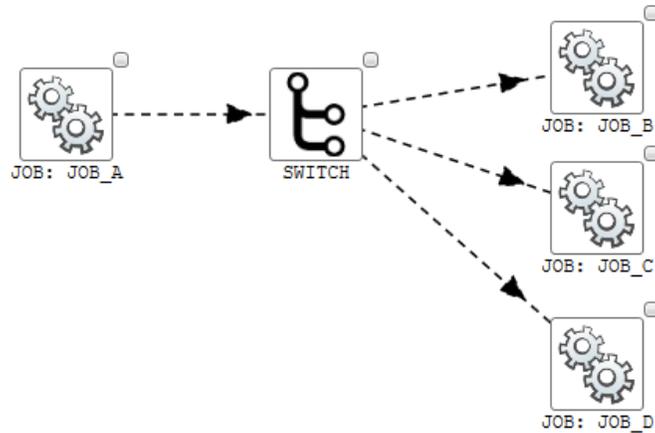
Branch success/failure paths based on input step state.

Example,

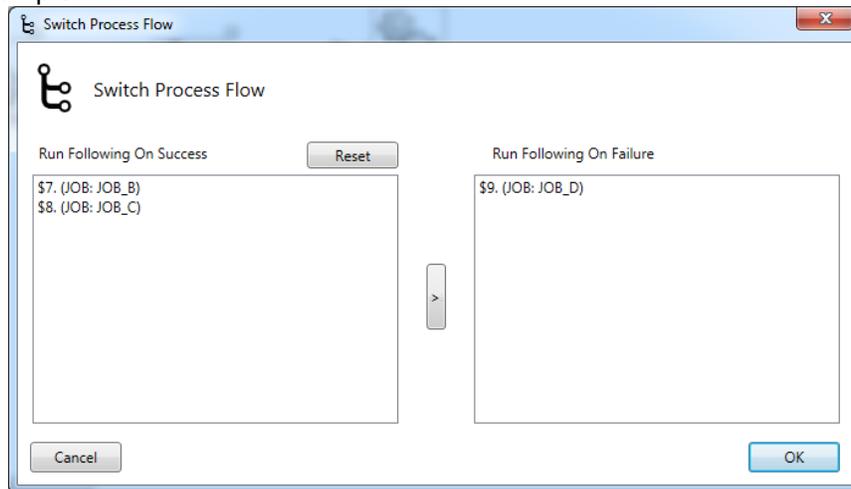
Sample dependency logic

1. Run JOB_A
2. If JOB_A succeeds Run JOB_B and JOB_C (Skip Failure Path)
3. If JOB_A fails Run JOB_D (Skip Success Path)

Workflow should look something like following.



Switch step property example.

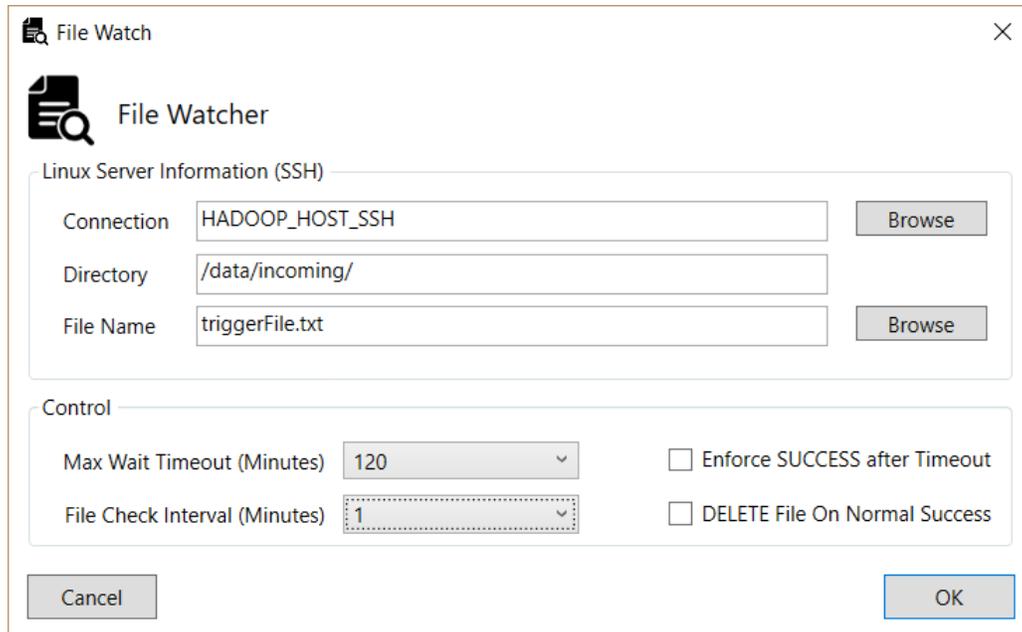


File Watch

Introduction

File Watch step waits until certain file on Unix/Linux system becomes available until timeout.

Example,



The screenshot shows a 'File Watcher' configuration window. It is divided into two main sections: 'Linux Server Information (SSH)' and 'Control'. In the 'Linux Server Information' section, there are three input fields: 'Connection' with the value 'HADOOP_HOST_SSH', 'Directory' with the value '/data/incoming/', and 'File Name' with the value 'triggerFile.txt'. Each field has a 'Browse' button to its right. The 'Control' section contains two dropdown menus: 'Max Wait Timeout (Minutes)' set to '120' and 'File Check Interval (Minutes)' set to '1'. To the right of these dropdowns are two checkboxes: 'Enforce SUCCESS after Timeout' and 'DELETE File On Normal Success', both of which are currently unchecked. At the bottom of the window are 'Cancel' and 'OK' buttons.

Linux Server Information

Linux server ssh credentials. Directory to be scanned and filename to watch for.

Max Wait Timeout

Wait for certain minutes until timeout has occurred.

File Check Interval

Interval to Check availability of file.

Enforce SUCCESS after Timeout

If option is checked step status will not fail after timeout. Leaving unchecked sets step status to Failed if file is not found and timeout has occurred.

Delete File On Normal Success

If checked, file gets deleted after setting status to success.

JDBC Watch

Introduction

JDBC Watch step waits until query returns certain value on a specified connection.

Example,

Watch JDBC Data State

Watch Data State Message (Polling) Connection: ADVENTUREWORKS2012

Watch Parameters

Match Values (Match on one of the following values)

43659

SQL Query (First column results are matched, Matching Limit = 100K Rows) Test Query

```
SELECT SalesOrderID from Sales.SalesOrderDetail where SalesOrderDetailID < 10
```

Control

Max Wait Timeout (Minutes) 120 Check Interval (Minutes) 1 Enforce SUCCESS after Timeout

Cancel OK

Match Values

Output to be matched against. True if one of the values match.

SQL Query

SQL query to collect output. State is success if any result tuple matches any specified match value (First row-column value).

Max Wait Timeout

Wait for certain minutes until timeout has occurred.

Check Interval

Interval to repeat query.

Enforce SUCCESS after Timeout

If option is checked, step status will not fail after timeout. Leaving unchecked sets step status to failed if match is not found.

Set Variable

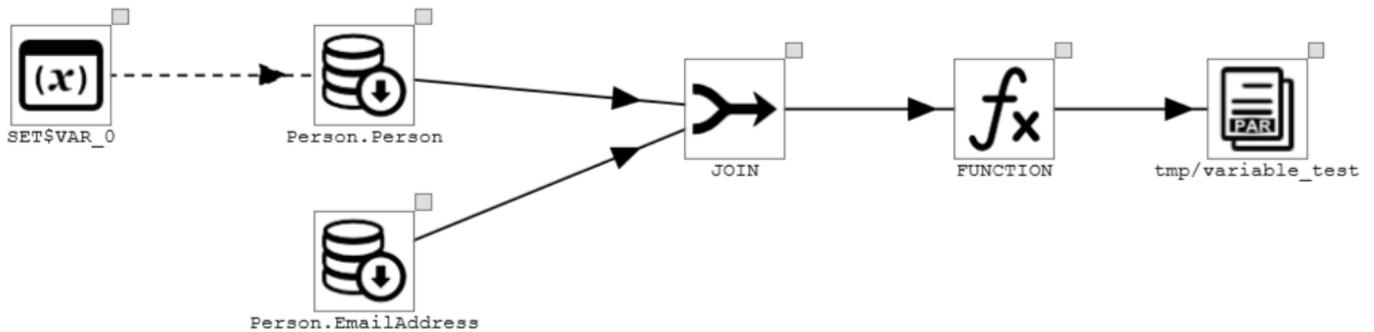
Introduction

The Set Variable step adds the ability to communicate between a Unix shell or an SQL database and an ELT Maestro program.

Each job is associated with a number of user variables, named \$VAR_0, \$VAR_1, \$VAR_2, ... (in addition to a number of system variables such as \$JOB_NAME, \$JOB_ID, etc.). The user variables are set in the Set Variable stage and accessed by various other stages. Variables facilitate communication between different parts of ELT Maestro programs. Because the Set Variable stage allows variables to have their values set by interaction with Unix shells and databases with which ELT Maestro can establish a connection, the Set Variable stage also facilitates communication between ELT Maestro processing Unix shells or databases of interest.

Example,

In this example, we use the Set Variable step to track the number of processes on one of the machines in the Spark cluster while the job is running and write that number to a column in an output table. Here we use a simple Join job, like the one used to demonstrate the Join command above. Next, we connect the Set Variable step to one of the OnStage input steps at the beginning of the program, as shown below:



Now open the Set Variable step:

Set Variable

Variable Name: \$VAR_0

Variable VALUE

Fixed SQL SHELL

Fixed Value

Static: HELLO

From Other Variables: [Dropdown]

Clear

SQL Output (First Tuple)

Connection: [Dropdown]

SQL Query: SELECT 'HELLO'

SHELL Script Output (First Word)

Connection: HADOOP1

Browse

Shell Script: ps -aef | wc -l

Cancel Test Value OK

In the Variable Name dropdown list, we have selected \$VAR_0 – in other words, \$VAR_0 is the particular variable we are setting.

From among the Fixed, SQL, and SHELL radio buttons, we have selected SHELL, indicating that the value that winds up in \$VAR_0 will come from a shell script.

HADOOP1 happens to be the name of the Unix machine we connect to. (That would likely be different in your case.) The shell script contains a short program to count the lines produced by the command **ps -aef**.

Note that we have also introduced a Function step between the Join step and the output file. The Function step allows us to retrieve the value of the variable and add it to the dataflow. Let's take a look inside the Function step:

Function

Input Output

Input Column(s)

BusinessEntityID

PersonType

NameStyle

Title

FirstName

MiddleName

LastName

Suffix

EmailPromotion

AdditionalContactInfo

Demographics

EmailAddress

Output Column(s) / Expression(s)

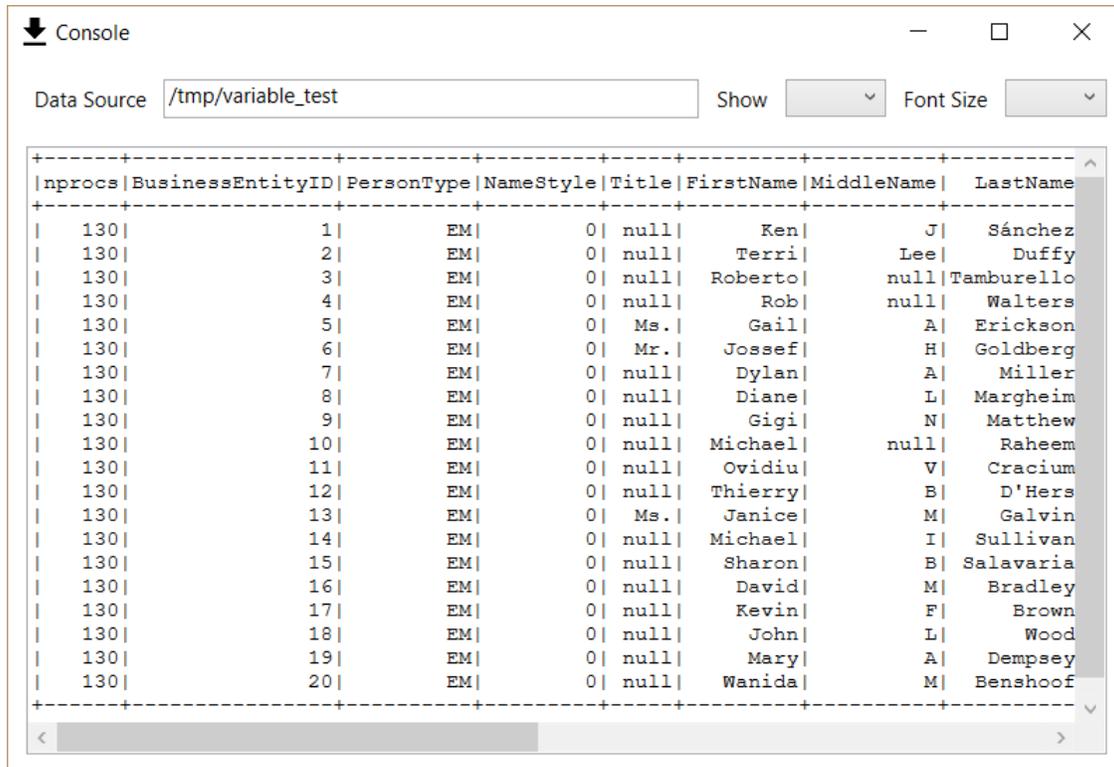
Add Expr Edit Delete Clear

Input Column(s)	Expression	Type
\$VAR_0	nprocs	IntegerType
BusinessEntityID	BusinessEntityID	IntegerType
PersonType	PersonType	StringType
NameStyle	NameStyle	IntegerType
Title	Title	StringType
FirstName	FirstName	StringType
MiddleName	MiddleName	StringType
LastName	LastName	StringType
Suffix	Suffix	StringType
EmailPromotion	EmailPromotion	IntegerType

Cancel OK

Here we see that we have added a new output column, named **nprocs**, of type integer, and set it to \$VAR_0.

After running the program, the results appear as follows:



Console window showing a table of data. The table has columns: nprocs, BusinessEntityID, PersonType, NameStyle, Title, FirstName, MiddleName, and LastName. The data shows 20 rows of employee information, with the first column (nprocs) consistently containing the value 130.

nprocs	BusinessEntityID	PersonType	NameStyle	Title	FirstName	MiddleName	LastName
130	1	EM	0	null	Ken	J	Sánchez
130	2	EM	0	null	Terri	Lee	Duffy
130	3	EM	0	null	Roberto	null	Tamburello
130	4	EM	0	null	Rob	null	Walters
130	5	EM	0	Ms.	Gail	A	Erickson
130	6	EM	0	Mr.	Jossef	H	Goldberg
130	7	EM	0	null	Dylan	A	Miller
130	8	EM	0	null	Diane	L	Margheim
130	9	EM	0	null	Gigi	N	Matthew
130	10	EM	0	null	Michael	null	Raheem
130	11	EM	0	null	Ovidiu	V	Cracium
130	12	EM	0	null	Thierry	B	D'Hers
130	13	EM	0	Ms.	Janice	M	Galvin
130	14	EM	0	null	Michael	I	Sullivan
130	15	EM	0	null	Sharon	B	Salavaria
130	16	EM	0	null	David	M	Bradley
130	17	EM	0	null	Kevin	F	Brown
130	18	EM	0	null	John	L	Wood
130	19	EM	0	null	Mary	A	Dempsey
130	20	EM	0	null	Wanida	M	Benshoof

That is, there were evidently 130 processes running on HADOOP1 as the Join job shown above ran.

Variable Name

Name of variable defined while creating job.

Variable Values

Fixed: Can be static value or copy from another variable.

SQL Output: Output of SQL query on specified connection is used to load variable value. First tuple (first row-column) is used from query result.

Shell Script: Output of shell script is used to load variable value. First word displayed on standard output is selected.

Note

All variable values are re-evaluated during runtime.

Watermark

Introduction

Sets workflow JOB or root job (batch) watermark from another variable. Watermark values can only be set by copying from pre-initialized variables (current job variables).

Example,

The screenshot shows a dialog box titled "Set Watermark". It contains a "Watermark Type" section with two radio buttons: "Batch" (selected) and "Job". Below this is a "Reset" button. The "Watermark Value" section has two dropdown menus: "High Value" with the value "\$VAR_0" and "Low Value" with the value "\$VAR_2". At the bottom of the dialog are "Cancel" and "OK" buttons.

Watermark Types

Batch: Root job watermark.
Job: Current job watermark.

Watermark Values

High Value: High watermark value for checked watermark type.
Low Value: Low watermark value for checked watermark type.

Note

When a workflow runs following watermark values are initialized automatically by the engine. The watermark values are captured from last successful run state.

\$BATCH_LOW_WATERMARK_VALUE
\$BATCH_HIGH_WATERMARK_VALUE
\$JOB_HIGH_WATERMARK_VALUE
\$JOB_LOW_WATERMARK_VALUE

Browse to **Variables and Watermark** section for more information.

SFTP

Introduction

SFTP step enables downloading files from remote UNIX/Linux servers. ELT Maestro connects using SSH protocol to retrieve files using secure channel. SFTP step can also utilize watermarks to enable downloading changed files.

Example,

The screenshot shows the Sftp configuration dialog box. It is divided into three main sections:

- Source Information (SFTP over SSH):** Includes fields for HostName (192.168.1.49), UserName (nz), Password (masked with dots), Directory (/nz/kit.7.1.0.3/log/postgres), and File/Pattern (pg.log.*). There are 'Test' and 'Browse' buttons.
- Target Information (Maestro Server):** Includes a Directory field (/tmp/postgres) with a 'Browse' button.
- Other Options:** Includes checkboxes for 'Use Watermark (\$JOB_LOW_WATERMARK_VALUE)', 'Auto-Clean' (checked), and 'Auto-Archive'.

At the bottom, there are 'Cancel' and 'OK' buttons.

Source Information

SSH Login: SSH credentials for SFTP server.
Directory: Source Directory
File/Pattern: File Name or Pattern. POSIX expression is used to evaluate file names.

Target Information

Directory: Directory path on ELT Maestro server.

Use Watermark Option

If this option is checked ensure that current workflow does not set Job Low Watermark Value. Watermark option utilizes job low watermark value to obtain only the changed files since last load. SFTP step automatically keeps track of latest file modified timestamp based on source server time zone and updates job low watermark value automatically.

Auto Clean Option

If this option is checked upon completion of root job (BATCH) the downloaded files are automatically deleted. Auto-Clean option is useful specially when freeing up disk resources on ELT Maestro server after loading them into database tables.

SFTP2S3 -Deprecated

Introduction

SFTP2S3 step enables downloading files from remote UNIX/Linux servers into one of the active agent systems and then uploading those files to S3 bucket. ELT Maestro connects using SSH protocol to retrieve files using secure channel. SFTP step can also utilize watermarks to enable downloading changed files. File(s) can be encrypted and/or compressed before uploading into S3 bucket. Client side AES256 bit encryption is default encryption algorithm.

Example,

The screenshot shows a configuration window for 'Sftp To AWS S3'. It has three main sections: 'Source Information (SFTP over SSH)', 'Target Information (S3 Storage)', and 'Other Options'.
- Source Information: Connection is 'HOST_44', Directory is '/data/outgoing', File/Pattern is 'orders*.psv', and Threads is set to 2.
- Target Information: S3 Connection is 'OREGON_S3_CONNECTION_Z', S3 Key Prefix is 'eltmaestro/data/incoming/', and File Part Size (MB) is 8.
- Other Options: 'Use Watermark (\$JOB_LOW_WATERMARK_VALUE)', 'Compress Level= 7', 'Encrypt File', and 'Auto-Clean' are all checked.

Parameters

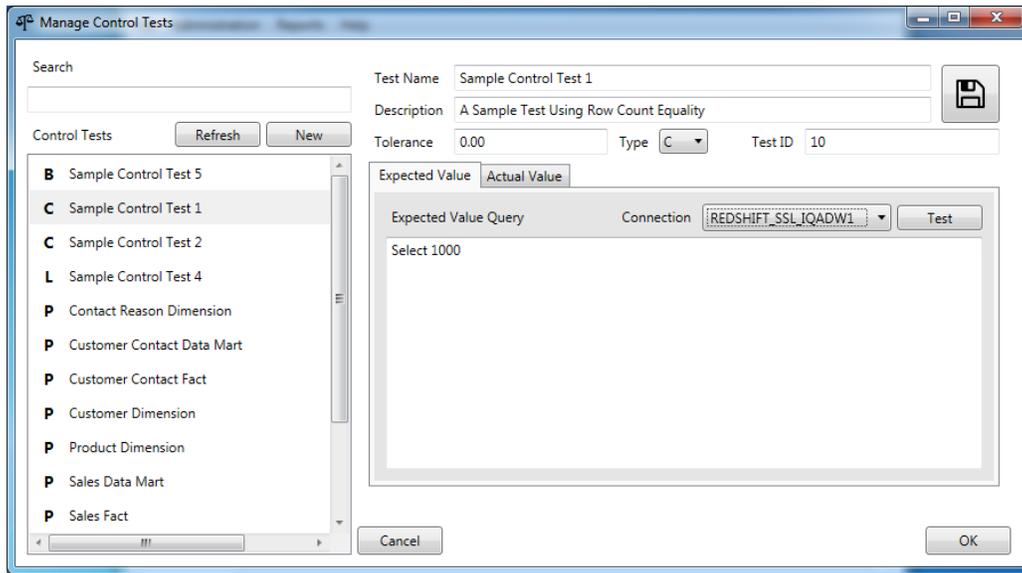
Property	Type	Info
Connection	Text	SSH Connection Name
Directory	Text	Source Directory
File / Pattern	Text	Filename or POSIX Pattern
Threads	Selection	Number of threads for parallel uploads. (Applies to pattern matched files)
File Part Size (MB)	Selection	S3 upload file partition size for larger files.
S3 Connection	Text	S3 Connection Name
S3 Key Prefix	Text	Key Prefix to append
Use Watermark	Check Box	Optional: Uses low watermark variable.
Compress	Selection	Optional: Uses bzip2 compression, 9 is highest compression level
Encrypt File	Check Box	Encrypts file before uploading using specified key configured on S3 Connection. Uses AES256 bit client side symmetric key.
Auto-Clean	Check Box	Deletes uploaded files from S3 upon job completion.

Control Test

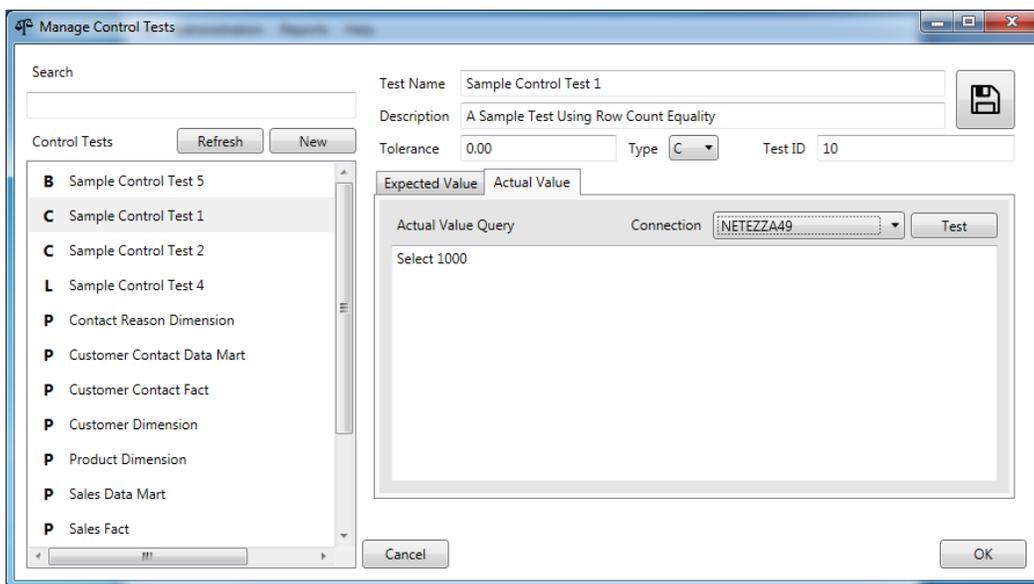
Introduction

Enables running data quality tests to measure the correctness or reasonableness of data as it is moved and transformed within or across workflows. A Control Test must be defined before it can be inserted into a workflow. The definition is performed using the Control Test Designer, which is accessed from the Administration Menu on the main ribbon.

To open control test editor, on the main window Click [Administration], Click [Metrics Configuration] then Click [Control Test]



After giving a name, an identifier, and defining the test type, tolerance, and the expected value query. We then define the connection and the actual value query as shown below. All normal control tests will require both an expected value query and an actual value query which are then compared to determine if the control test passed or failed. Control Test Type 'M' which indicates we are performing a measurement rather than a control test, is used to probe and report on queryable system parameters we may wish to know about and run trending analysis on. Example include measuring database size growth over time, capturing end-user counts, or tracing the number of reports run.



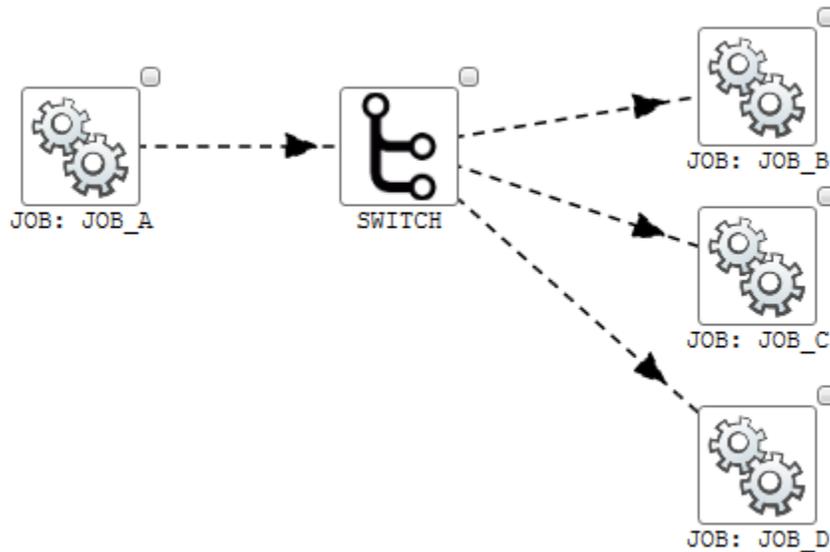
We can now save the Control Test and then call it within a workflow. The illustration below shows how it is selected from within the workflow designer.

Job

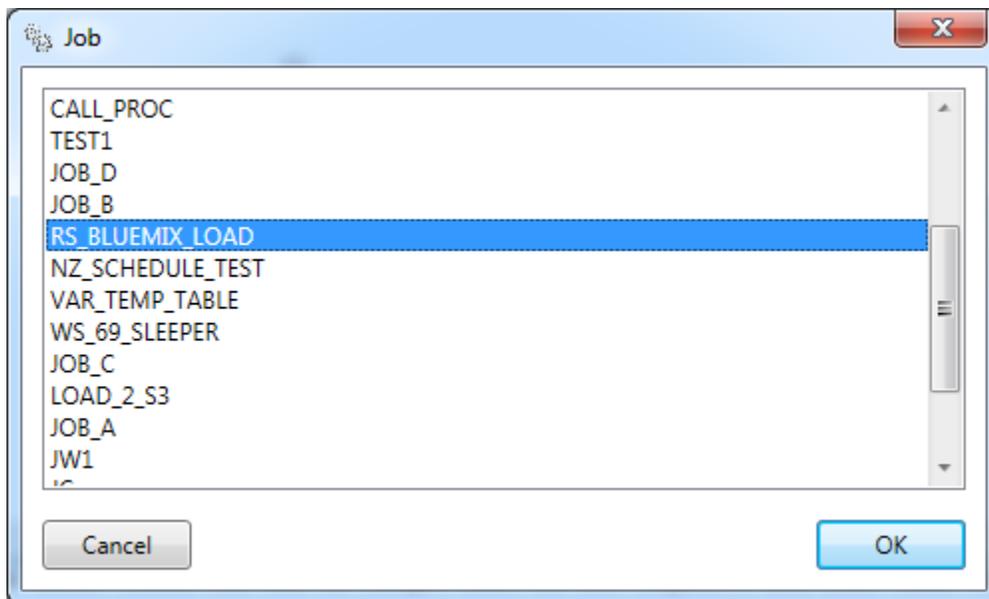
Introduction

Allows current workflow to execute deployed workflow. Job can be used with Switch to design success and recovery workflow path as well.

Example Workflow Implementation,



Example,



Job

Job Step executes selected deployed job.

Note

ELT Maestro engine can only execute one instance of a job in a workflow to avoid execution recursion.

File Loader [Snowflake]

Introduction

File loader step allows loading flat files into existing table. The table structure must match data file structure. Output of file loader step should point to physical table. File Loader step can load files from Onstage source, SFTP source or files located on ELT Maestro server.

Figure: File Loader Step

Columns List	Column Name	Column Type	Key
	ISN	BIGINT	<input type="checkbox"/>
	TEXT_LENGTH	BIGINT	<input type="checkbox"/>
	TARGET_ID	VARCHAR(68)	<input type="checkbox"/>
	ORIGINAL_TARGET	VARCHAR(68)	<input type="checkbox"/>
	SOURCE_ID	VARCHAR(68)	<input type="checkbox"/>
	SOURCE_MSC_ADDRESS	VARCHAR(68)	<input type="checkbox"/>
	SEND_MODE	BIGINT	<input type="checkbox"/>

Requirements

- A table must exist on database that matches metadata with file so a known structure can be referenced
- File path must be specified if loading from files located on ELT Maestro server
- Optionally based on file construct you must alter [Load Query Options], refer to Netezza external table options for customizing load options
- An External File Stage must be configured
- File Format must be configured

Loader Configuration Steps

- Create a table on Snowflake database that matches with file structure
- Reference whether file is provided by sftp, onstage or locally present
- Modify external table load options without changing \$ variable references
- Attach output to table step as shown on figure below.

Figure: File Loader to Table



Onstage Group [Snowflake]

Introduction

Onstage Group also known as schema loader is used to load multiple databases tables from a source jdbc connection. This step is a database migration utility that can generate objects on target system like Snowflake and loads data from source systems. Incremental loads can be achieved by utilizing watermark columns.

Figure: Onstage Group Loader [Schema Tab]

The screenshot shows the 'Onstage Schema Loader for Snowflake' window with the 'Schema' tab selected. The window is divided into several sections:

- Source:** Connection (NETEZZA), Catalog (GTT_STAGE), Schema (ADMIN). A 'Browse' button is next to the Schema field.
- Target:** Catalog (ATNI_PROD), Schema (GTT_REPORTING). A 'Browse' button is next to the Schema field.
- Configurations:**
 - Initial Refresh Partitions: 128
 - Yearly Data Growth Factor (Optional): 1.05
 - File Format: "ATNI_PROD"."ELTMAESTRO"."INTEGRATOR_ONSTAGE_FORMAT". A 'Browse' button is next to the field.
 - File Stage: "ATNI_PROD"."ELTMAESTRO"."ELTM_AZURE_STAGE". A 'Browse' button is next to the field.

At the bottom of the window, there are 'Cancel' and 'Save' buttons.

Configuration

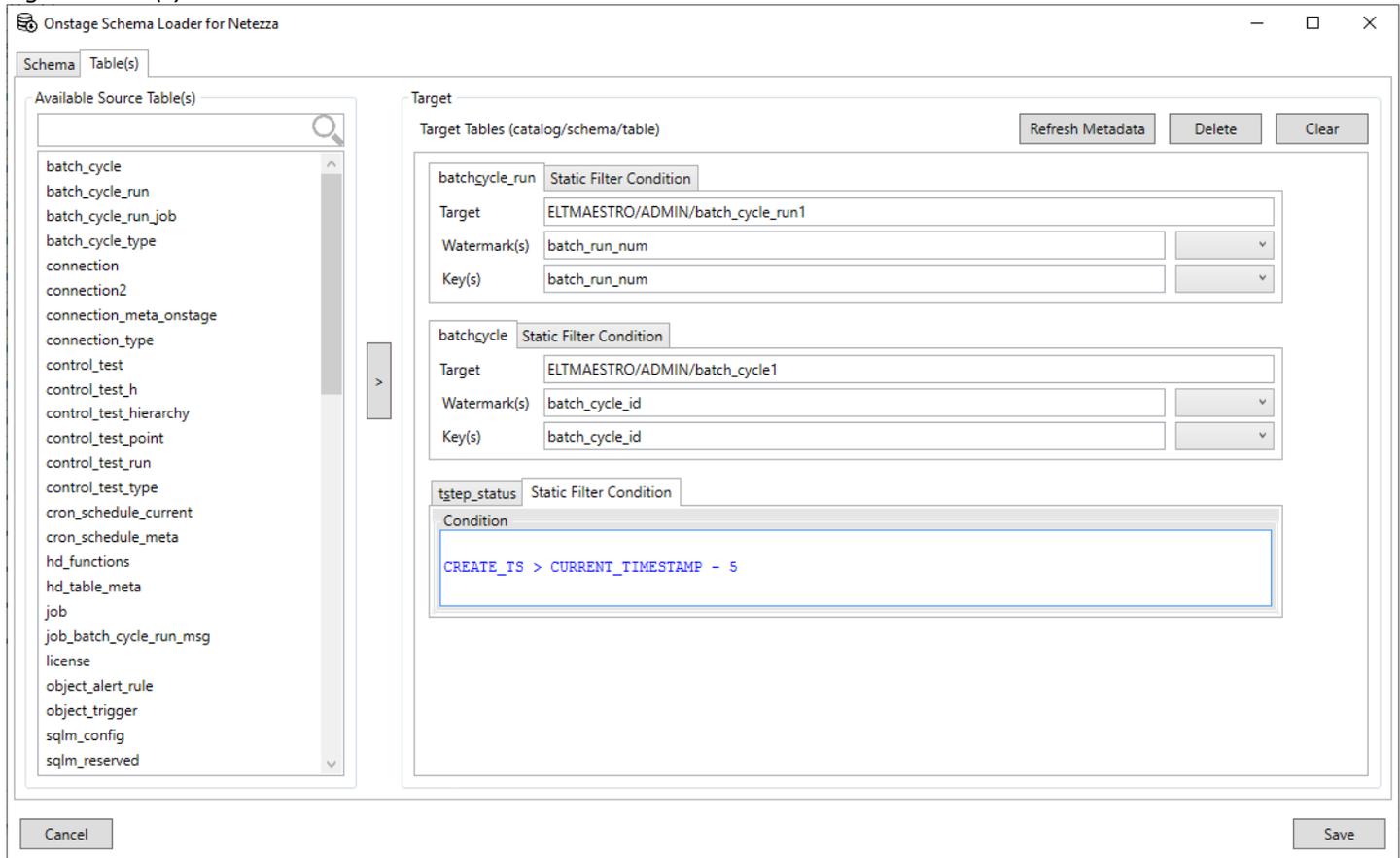
- Browse to [Select] source connection, catalog and schema to migrate tables from
- Browse to [Select] target catalog and schema to migrate tables to
- Select [Initial Load Partitions] to bucket initial load sql script into multiple conditions
- Select [Initial Load Threads] to specify parallelism factor on partitions
- Select [File Format] and [File Stage]
- Optionally if you set [Run Count Metrics] to true, ELT Maestro automatically runs source/target row counts before and after load has been completed.

Example,

If you set initial load partitions to 16 and load threads to 4 then ELT Maestro will generate 16 range based SQL queries to execute on source system and executes those 16 queries 4 at a time, the load is also performed at parallelism of 4. Setting initial load partitions is best if you are loading a very large tables.

Note: A numeric or timestamp based watermark column is always required to partition the data

Figure: Table(s)



Adding Tables

- Select tables on left panel and Click [>] button to add to list on right panel
- Click [Refresh Metadata] button to load metadata information after all required tables have been added

Delta Configuration (Option 1 if you know delta columns)

- For each table, select [watermark] column, you can select multiple watermark columns for delta load
- Specify [key] columns so that updates can be upserted. If keys are not specified the loader will only perform inserts

Reach-Back or Manual SQL Configuration (Option 2 if you don't know delta columns)

- Modify [Static Filter Condition] and add a filter expression. As shown on above diagram for third table a static filter is added that goes back 4 days to obtain records.

Note: Object path is referenced with path format of catalog/schema/table. If you are going to change table names on target please use this format.

Table [Snowflake]

Introduction

Table step can read from or write to referenced table.

Figure: Table Step

Column Name	Data Type	Cluster	Key	Ident
BATCH_RUN_NUM	NUMBER(38,0)	<input checked="" type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
BATCH_CYCLE_ID	NUMBER(38,0)	<input type="checkbox"/>	<input checked="" type="checkbox"/>	<input type="checkbox"/>
BATCH_RUN_START_TS	TIMESTAMP_NTZ(9)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_RUN_END_TS	TIMESTAMP_NTZ(9)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_STATUS_CD	VARCHAR(16)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_CYCLE_DATA_VALIDITY_TS	TIMESTAMP_NTZ(9)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_CYCLE_HIGHWATER_MARK	VARCHAR(60)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_CYCLE_LOWWATER_MARK	VARCHAR(60)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
COMPLETED	BOOLEAN	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ERROR_COUNT	NUMBER(38,0)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CREATE_TS	TIMESTAMP_NTZ(9)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Construct

1. [Existing] Reference existing table and metadata from database schema
2. [Create] Create table if it does not exist
3. [Temp] Write output to a temp table, then drop it after workflow execution is completed

Load Options

1. [Truncate] Truncate table before inserting data
2. [Run Statistics] Run GENERATE STATISTICS command on table after data has been inserted or updated
3. [Run Vacuum] Run GROOM command on table after data has been inserted or updated
4. [Upsert] Update + Insert data on table by detecting key column reference

Column Properties

1. [Cluster] Cluster rows based on check marked column
2. [Key] Mark column as a logical key (required for upsert mode)
3. [Ident] Mark column as identity/auto incrementing column

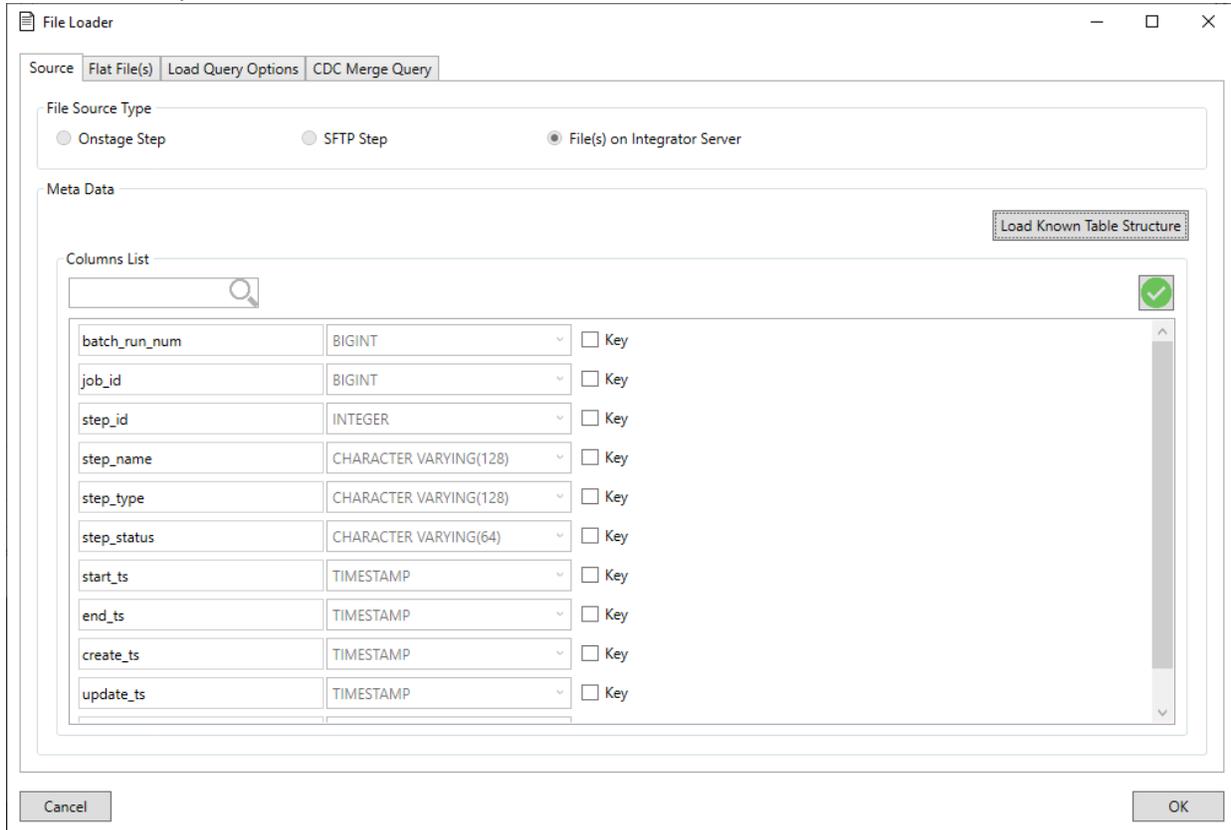
You can only add/remove columns for construct type [Create] or [Temp] or set Ident flag to true.

File Loader [Netezza]

Introduction

File loader step allows loading flat files into existing table. The table structure must match data file structure. Output of file loader step should point to physical table. File Loader step can load files from Onstage source, SFTP source or files located on ELT Maestro server.

Figure: File Loader Step



Requirements

- A table must exist on database that matches metadata with file so a known structure can be referenced
- File path must be specified if loading from files located on ELT Maestro server
- Optionally based on file construct you must alter [Load Query Options], refer to Netezza external table options for customizing load options

Loader Configuration Steps

- Create a table on Netezza database that matches with file structure
- Reference whether file is provided by sftp, onstage or locally present
- Modify external table load options without changing \$ variable references
- Attach output to table step as shown on figure below.

Figure: File Loader to Table



Onstage Group [Netezza]

Introduction

Onstage Group also known as schema loader is used to load multiple databases tables from a source jdbc connection. This step is a database migration utility that can generate objects on target system like Netezza and loads data from source systems. Incremental loads can be achieved by utilizing watermark columns.

Figure: Onstage Group Loader [Schema Tab]

The screenshot shows the 'Onstage Schema Loader for Netezza' application window. The 'Schema' tab is selected. The 'Source' section has 'Connection' set to 'POSTGRES', 'Catalog' to 'sqlmaestro', and 'Schema' to 'public'. The 'Target' section has 'Catalog' set to 'ELTMAESTRO' and 'Schema' to 'ADMIN'. The 'Load Configurations' section includes: 'Initial Load Partitions' (4), 'Initial Load Threads' (2), 'Attributes Quoting' (True), 'Run Count Metrics' (True), and 'Data Growth Rate (Beta)' (1.05). The 'Additional Configuration' section is currently empty and marked as 'not available'. 'Cancel' and 'Save' buttons are at the bottom.

Configuration

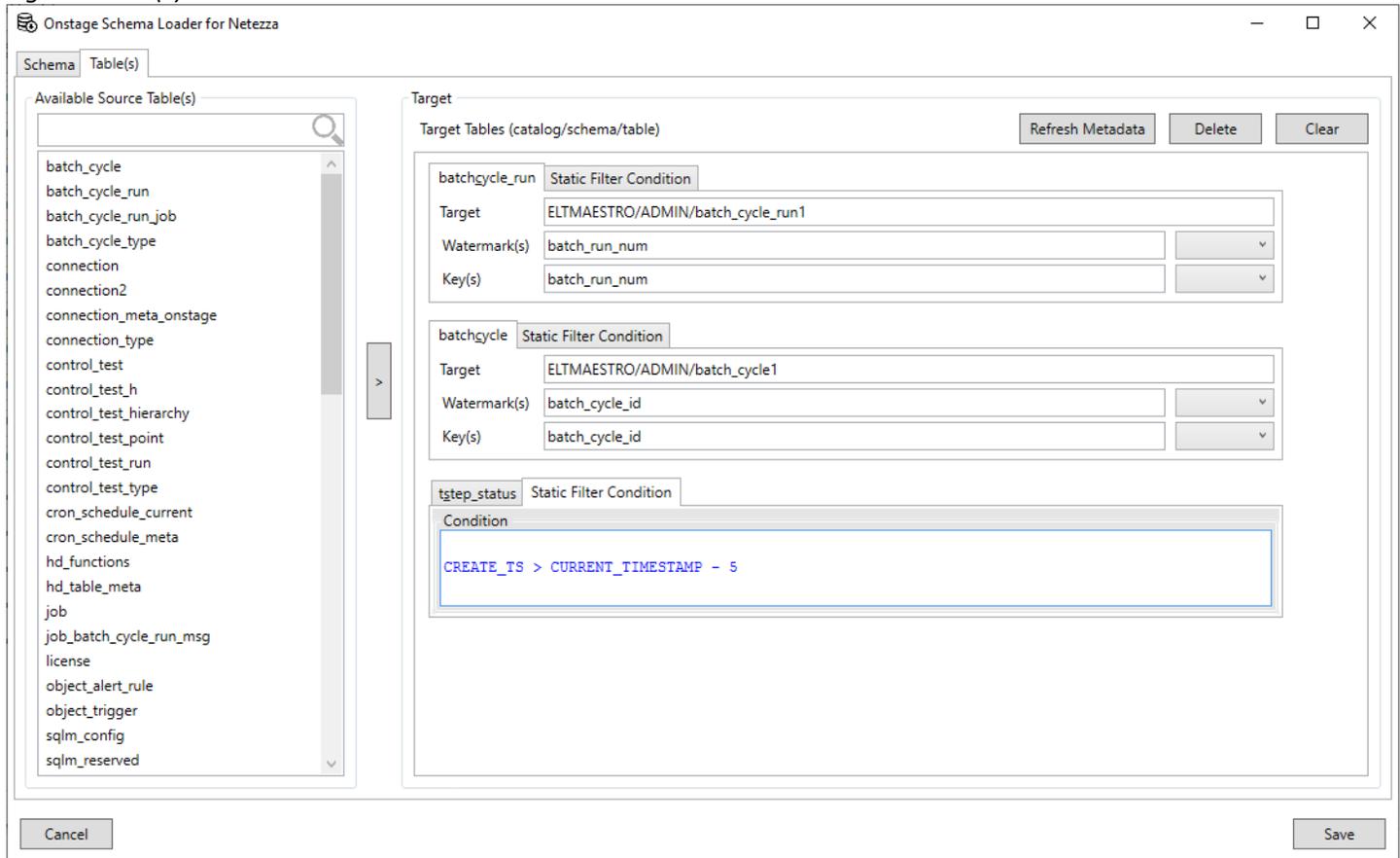
- Browse to [Select] source connection, catalog and schema to migrate tables from
- Browse to [Select] target catalog and schema to migrate tables to
- Select [Initial Load Partitions] to bucket initial load sql script into multiple conditions
- Select [Initial Load Threads] to specify parallelism factor on partitions
- Optionally if you set [Run Count Metrics] to true, ELT Maestro automatically runs source/target row counts before and after load has been completed.

Example,

If you set initial load partitions to 16 and load threads to 4 then ELT Maestro will generate 16 range based SQL queries to execute on source system and executes those 16 queries 4 at a time, the load is also performed at parallelism of 4. Setting initial load partitions is best if you are loading a very large tables.

Note: A numeric or timestamp based watermark column is always required to partition the data

Figure: Table(s)



Adding Tables

- Select tables on left panel and Click [>] button to add to list on right panel
- Click [Refresh Metadata] button to load metadata information after all required tables have been added

Delta Configuration (Option 1 if you know delta columns)

- For each table, select [watermark] column, you can select multiple watermark columns for delta load
- Specify [key] columns so that updates can be upserted. If keys are not specified the loader will only perform inserts

Reach-Back or Manual SQL Configuration (Option 2 if you don't know delta columns)

- Modify [Static Filter Condition] and add a filter expression. As shown on above diagram for third table a static filter is added that goes back 4 days to obtain records.

Note: Object path is referenced with path format of catalog/schema/table. If you are going to change table names on target (Netezza) please use this format.

Table [Netezza]

Introduction

Table step can read from or write to referenced table.

Figure: Table Step

Table

Construct

Existing Create Temp

Load Options

Truncate Run Statistics
 Run Vacuum Upsert

Table

Database: TESTDB
Schema: ADMIN
Table: RPT_BATCH_STATISTICS

Columns List

Column Name	Data Type	Dist	Sort	Key
BATCH_CYCLE_ID	BIGINT	<input type="checkbox"/>	<input type="checkbox"/>	<input checked="" type="checkbox"/>
BATCH_CYCLE_NM	CHARACTER VARYING(255)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
BATCH_STATUS_CD	CHARACTER VARYING(12)	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
RUN_COUNT	INTEGER	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Construct

4. [Existing] Reference existing table and metadata from database schema
5. [Create] Create table if it does not exist
6. [Temp] Write output to a temp table, then drop it after workflow execution is completed

Load Options

5. [Truncate] Truncate table before inserting data
6. [Run Statistics] Run GENERATE STATISTICS command on table after data has been inserted or updated
7. [Run Vacuum] Run GROOM command on table after data has been inserted or updated
8. [Upsert] Update + Insert data on table by detecting key column reference

Column Properties

4. [Dist] Distribute data based on check marked column (DISTRIBUTE ON)
5. [Sort] Organize/Cluster data based on check marked column (ORGANIZE ON)
6. [Key] Mark column as a logical key (required for upsert mode)

You can only add/remove columns for construct type [Create] or [Temp]

Onstage Group [Spark SQL]

Introduction

Onstage Group also known as schema loader is used to load multiple databases tables from a source jdbc connection. This step is a database migration utility that can generate objects on target system like Hadoop and loads data from source systems. Incremental loads can be achieved by utilizing watermark columns.

Figure: Onstage Group Loader [Schema Tab]

The screenshot shows the 'Onstage Schema Loader for Spark' application window. The 'Schema' tab is selected. The 'Source' section includes a 'Connection' dropdown set to 'POSTGRES', a 'Catalog' text field with 'sqlmaestro', and a 'Schema' text field with 'public' and a 'Browse' button. The 'Target' section includes a 'Catalog' text field with 'none' and a 'Folder' text field with '/tmp' and a 'Browse' button. The 'Load Configurations' section has five dropdowns: 'Initial Load Partitions' (4), 'Initial Load Threads' (2), 'Attributes Quoting' (True), 'Run Count Metrics' (False), and 'Data Growth Rate (Beta)' (1.05). The 'Additional Configuration [not available]' section is empty. 'Cancel' and 'Save' buttons are at the bottom.

Configuration

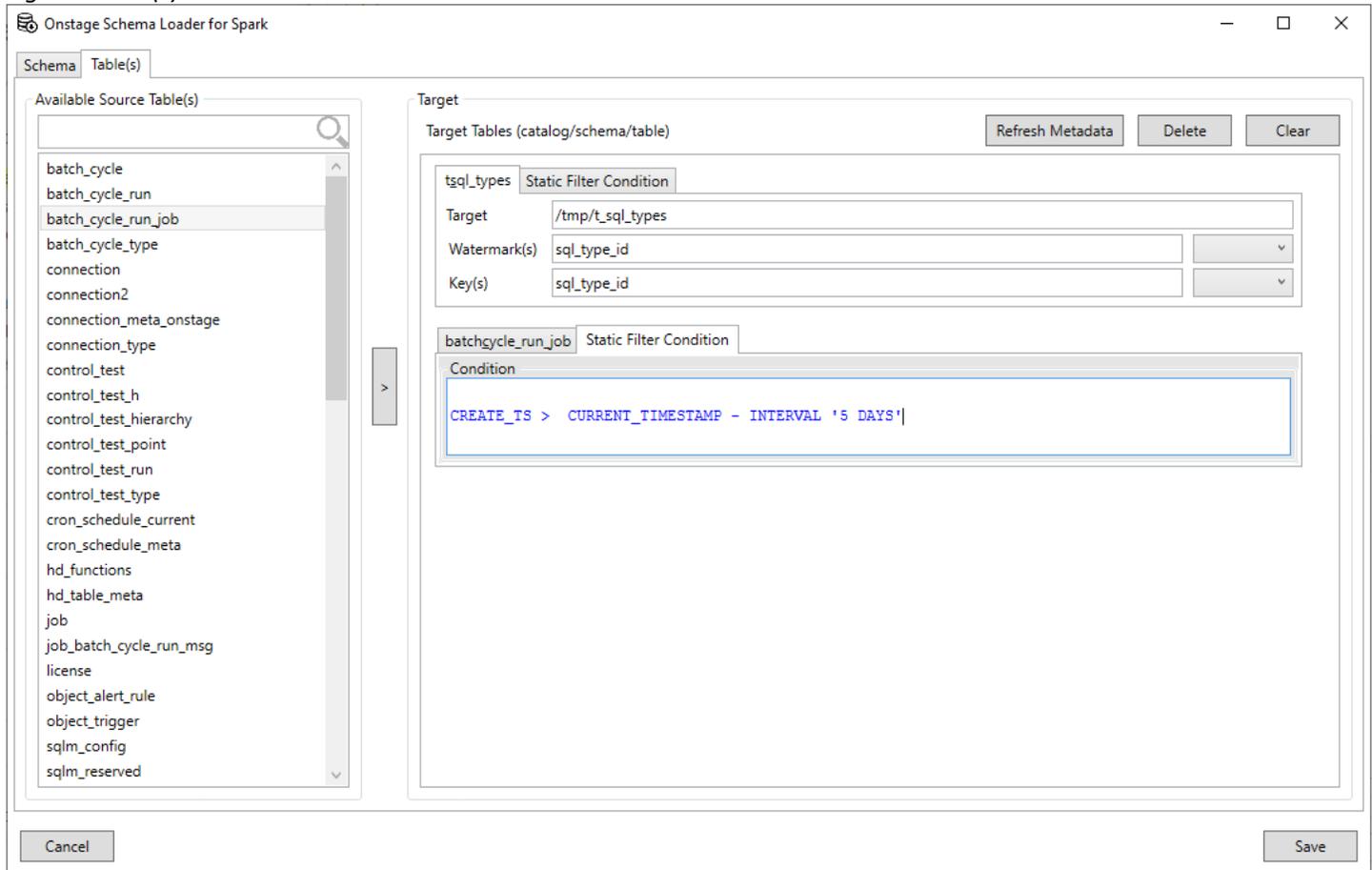
- Browse to [Select] source connection, catalog and schema to migrate tables from
- Browse to [Select] target hdfs path and schema to migrate tables to
- Select [Initial Load Partitions] to bucket initial load sql script into multiple conditions
- Select [Initial Load Threads] to specify parallelism factor on partitions
- Optionally if you set [Run Count Metrics] to true, ELT Maestro automatically runs source/target row counts before and after load has been completed.

Example,

If you set initial load partitions to 16 and load threads to 4 then ELT Maestro will generate 16 range based SQL queries to execute on source system and executes those 16 queries 4 at a time, the load is also performed at parallelism of 4. Setting initial load partitions is best if you are loading a very large tables.

Note: A numeric or timestamp based watermark column is always required to partition the data

Figure: Table(s)



Adding Tables

- Select tables on left panel and Click [>] button to add to list on right panel
- Click [Refresh Metadata] button to load metadata information after all required tables have been added

Delta Configuration (Option 1 if you know delta columns)

- For each table, select [watermark] column, you can select multiple watermark columns for delta load
- Specify [key] columns so that updates can be upserted. If keys are not specified the loader will only perform inserts

Reach-Back or Manual SQL Configuration (Option 2 if you don't know delta columns)

- Modify [Static Filter Condition] and add a filter expression. As shown on above diagram for third table a static filter is added that goes back 4 days to obtain records.

Note: Object path is referenced with HDFS path format without url prefix.

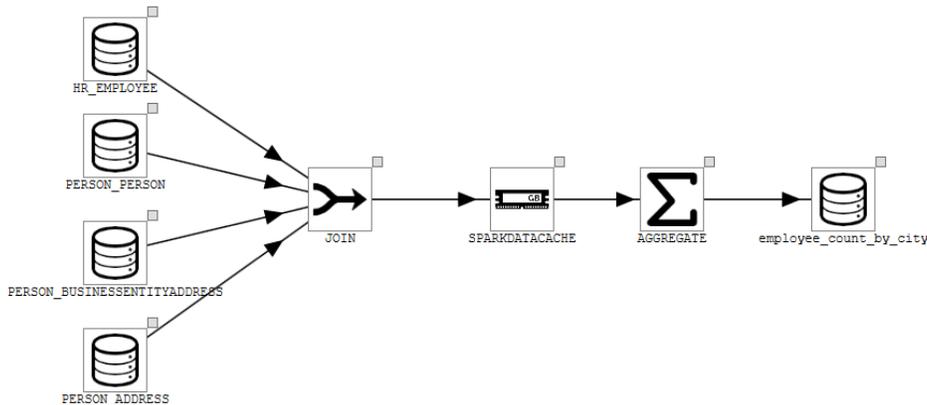
Data Frame [Spark SQL]

Introduction

Data Frame step reads file(s) or folder on HDFS and converts into data frame object. Additionally this step also creates a temporary global view for SQL operation. This step can load file types such as Parquet, Orc, Csv, Json and Text and convert into structured data frame object. This step is very similar to table step.

Consider following workflow

Figure: Data Frame on a workflow



The 4 data-frame steps on left are reader steps and the last one (employee_count_by_city) is a writer step. You can tell this by looking at the connecting arrow(s). The tail represents read and head represents write.

Figure: Data Frame

Spark Data Frame/File

Data Source: CSV File Options | JSON Options | Output Mode

File Format: Parquet

Compression: snappy

Write Options: Truncate, Append, Upsert-Append

File Parts (Coalesce): auto

Data Object Path: /adventureworks/HR_EMPLOYEE

Columns List:

Column Name	Column Type	Dist	Sort	PK/FK
HIREDATE	Timestamp	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SALARIEDFLAG	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
VACATIONHOURS	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
SICKLEAVEHOURS	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CURRENTFLAG	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ROWGUID	String	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
MODIFIEDDATE	Timestamp	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
DATASLICEID	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
CREATExID	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>
ELTM_TXID	Integer	<input type="checkbox"/>	<input type="checkbox"/>	<input type="checkbox"/>

Buttons: Cancel, OK

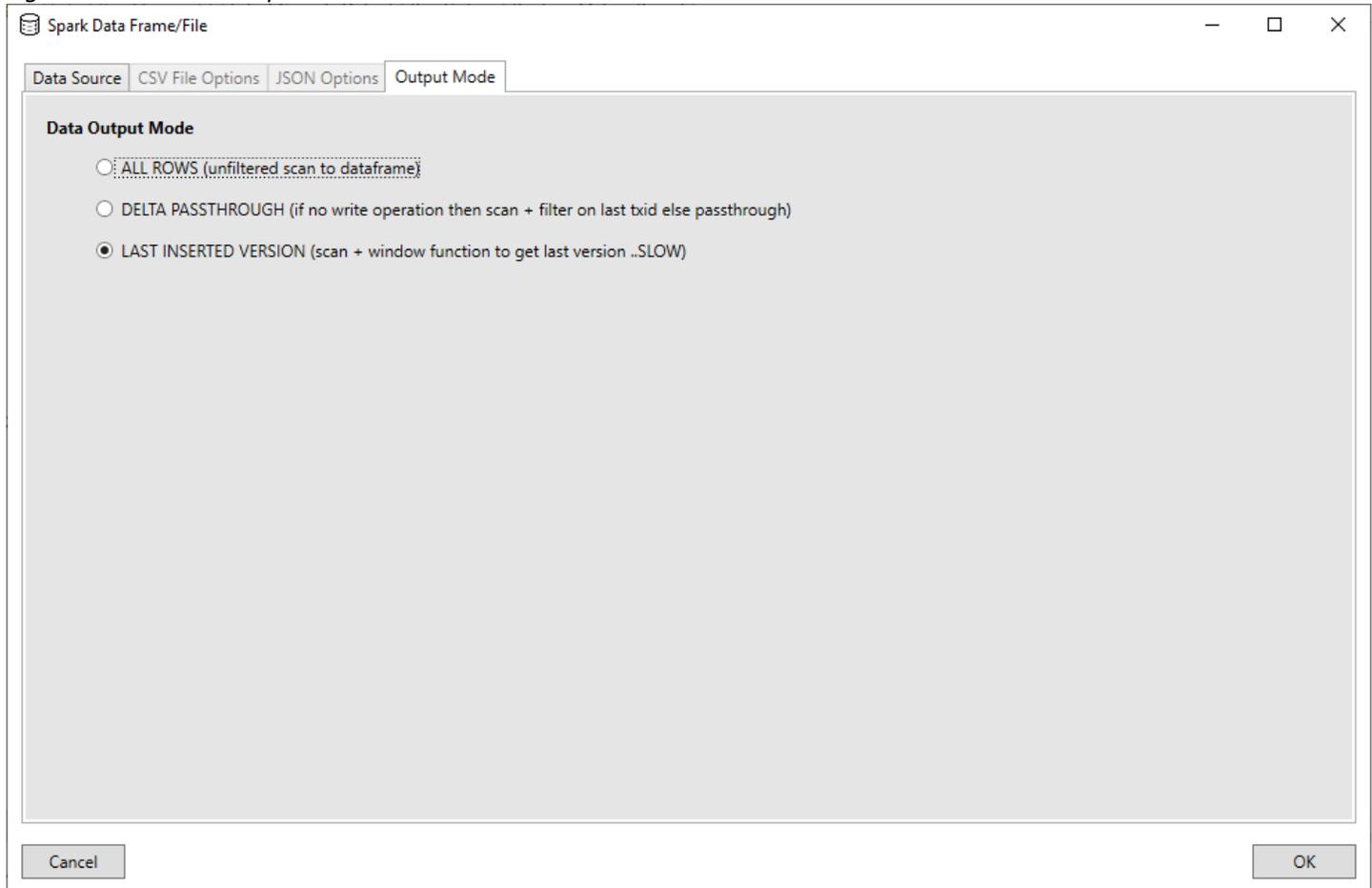
Data Frame Criteria

- If supplied path exists, you can read or append/write
- If supplied path does not exist, you can append/write

Configurations

- [File Format] can be Parquet, Orc, Csv, Json or Text and supported compression for these formats. For CSV or Json type please verify additional format configuration.
- [Compression] depends on supported file format by spark application.
- [Write Option] can be truncate which over-writes target path, append which keeps creating new files or upsert-append which keeps latest version of keys.
- [File Parts] is the number of files to write/create during write operation.

Figure: Data Frame output mode



Output Mode

- [All Rows] option provides all records available on specified path
- [Delta Passthrough] option provides records inserted by current workflow
- [Last Inserted Version] option provides latest record, for this to work you need to specify key columns for last version calculation.

Not implemented (YET)

- Enforcing distribution and sort columns.

Local File [Spark SQL]

Introduction

- Local File step is like data frame step with a condition that it is only a READER for file(s) or folder located on runtime ELT Maestro master or slave server.
- You cannot write to path referenced
- You can only run spark job on non-cluster mode (because you are referencing local file)

Spark Data Cache / Persistence [Spark SQL]

Introduction

Spark Data Cache allows caching data frame into disk, memory or both with options. This step does exactly what a spark persist function does and has same output as input.

Figure: Spark Data Cache

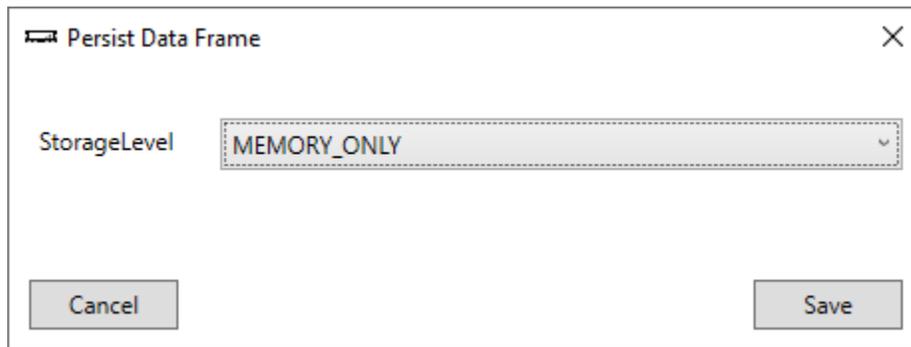
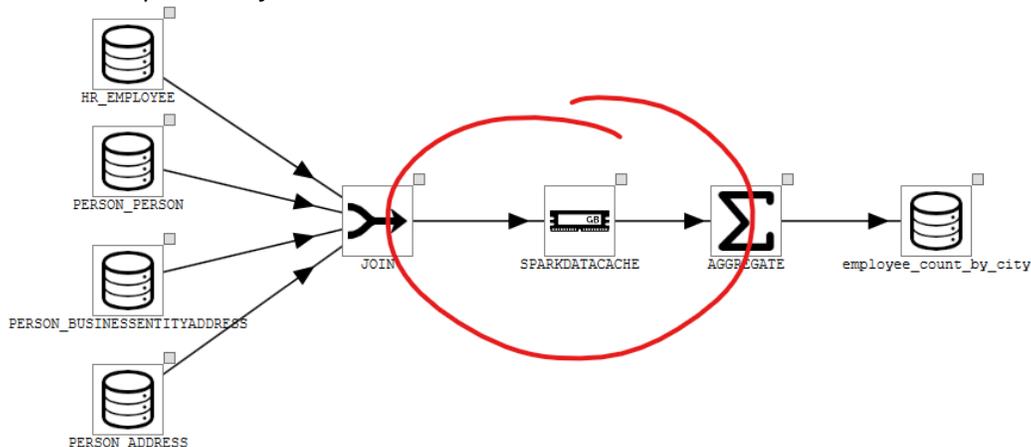


Figure: Spark Data Cache step on workflow



Available Cache/Persistence Options

- DISK_ONLY
- DISK_ONLY_2
- MEMORY_AND_DISK
- MEMORY_AND_DISK_2
- MEMORY_AND_DISK_SER
- MEMORY_AND_DISK_SER_2
- MEMORY_ONLY
- MEMORY_ONLY_2
- MEMORY_ONLY_SER
- MEMORY_ONLY_SER_2

Dedupe

Introduction

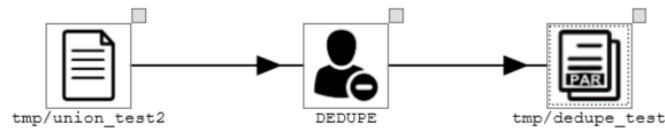
The Dedupe step removes duplicate rows from a dataset.

Example,

There are no properties to set.



Consider the following job:



Input	Output (after dedupe)
<pre>+-----+ MEMBER_ID MEMBER_NAME +-----+ 1001 aaaa 1002 bbbb 1003 cccc 1004 dddd 1021 uuuu 1022 vvvv 1001 aaaa 1002 bbbb 1003 cccc 1004 dddd 1021 uuuu 1022 vvvv +-----+</pre>	<pre>+-----+ MEMBER_ID MEMBER_NAME +-----+ 1002 bbbb 1021 uuuu 1022 vvvv 1004 dddd 1001 aaaa 1003 cccc +-----+</pre>

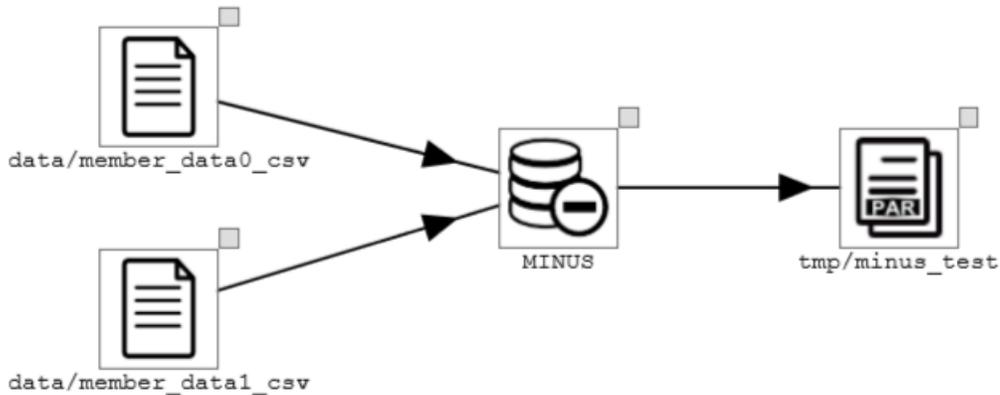
Minus

Introduction

The Minus step subtracts the contents of one dataset from another.

Example,

Consider the job shown below:



In this job, the contents of the table MEMBER_NAMES_2 are subtracted from the contents of the table MEMBER_NAMES. The column metadata for both inputs must be the same. In the properties window, you specify which dataset is the minuend (i.e. Source(A)) and which dataset is the subtrahend (i.e. Source(B)).

Minus

Source(A)-Source(B)

Source(A)

All Input Columns

- \$8.MEMBER_ID
- \$8.MEMBER_NAME**
- \$9.MEMBER_ID
- \$9.MEMBER_NAME

Output Columns

-
-

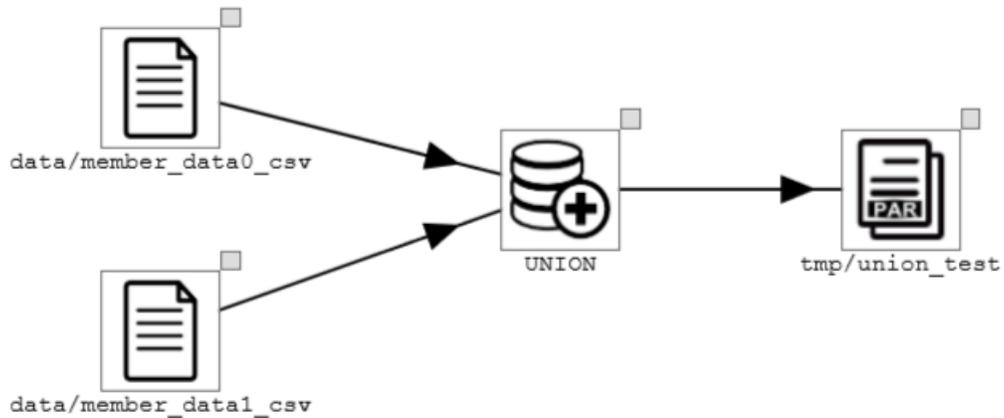
Buttons: Arrange Columns, Delete, Refresh, Cancel, OK

Union

Introduction

The Union step combines two datasets. The datasets have to have the same column metadata.

Example,



Here, the Union step is used to combine the contents of two flat files, and the result is loaded to a table.

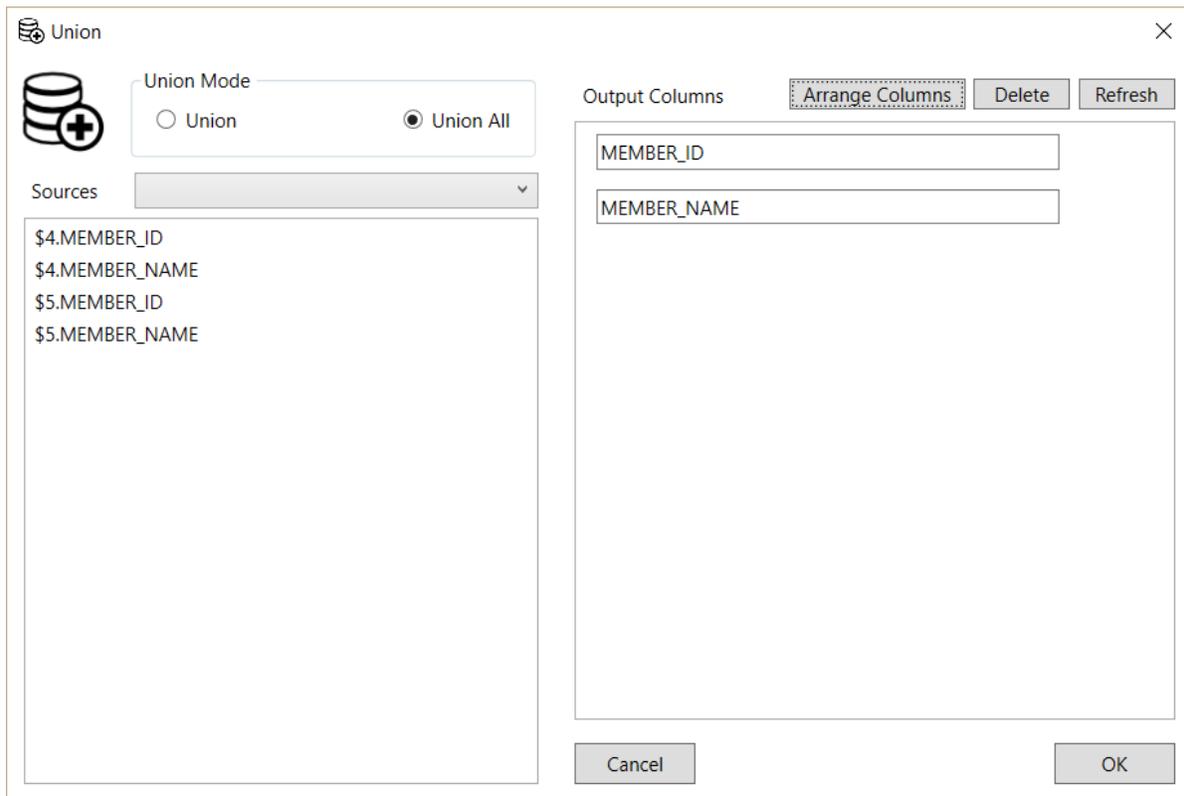
The contents of the first flat file is

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1005	eeee
1006	ffff

and the contents of the second flat file is

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1021	uuuu
1022	vvvv

The Union step properties window is set up as follows:



In the Sources drop-down list choose ALL.

The output is:

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	ccc
1004	ddd
1021	uuuu
1022	vvvv
1001	aaaa
1002	bbbb
1003	ccc
1004	ddd
1005	eeee
1006	ffff

Only the Union All option is available in ELT Maestro for Spark. (In other ELT Maestro editions, the Union option would remove duplicates.)

All rows in Source(A) matching any row in Source(B) will be removed from the result, regardless of how many duplicates there are. For example if the inputs are as follows:

Source(A):

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1005	eeee
1006	ffff

Source(B):

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1021	uuuu
1022	vvvv

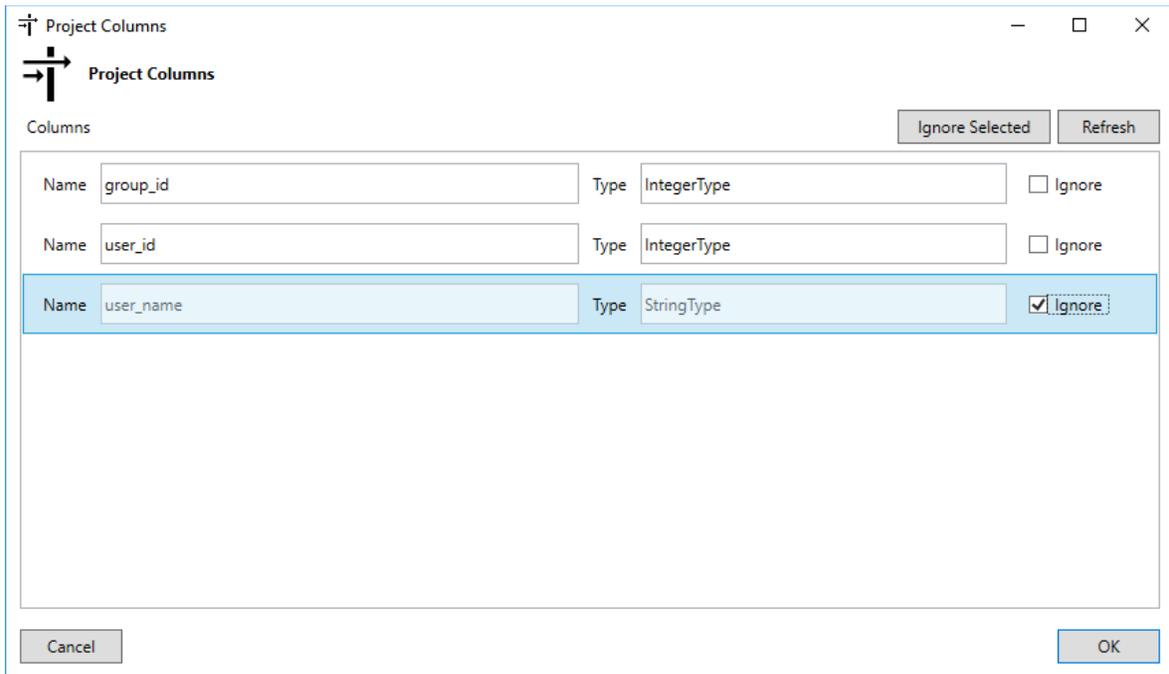
The output will be:

MEMBER_ID	MEMBER_NAME
1005	eeee
1006	ffff

Projection (Deprecated)

Introduction

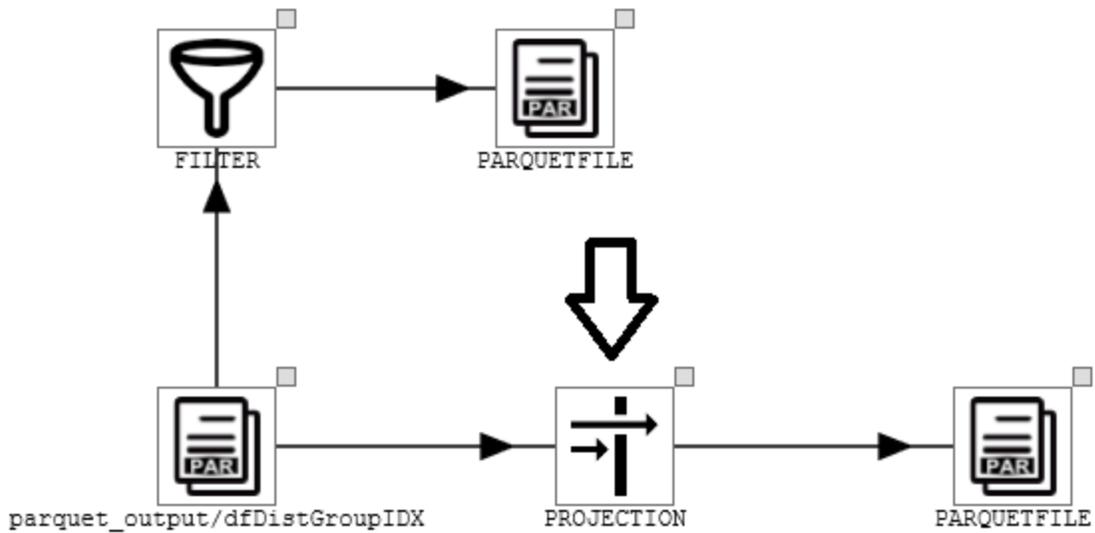
The Project Columns step allows you to drop some of the columns from a dataset. It has a very straightforward interface. This step is used to limit columns.



Example,

Simply check Ignore on the columns you wish to drop.

The example above is taken from the following job:



Aggregate

Introduction

The Aggregate step performs aggregations.

Example,

Suppose the table MEMBER_SCORES contains the following data:

MEMBER_ID	PLUSES	MINUSES
1002	4	5
1002	5	6
1002	6	8
1003	6	6
1003	2	4
1003	4	7
1003	1	6
1006	4	3
1007	8	2

We will create a job to aggregate by MEMBER_ID, summing all of the PLUSES with the same MEMBER_ID and averaging all of the MINUSES with the same MEMBER_ID. The job looks like this:

The properties window for the Aggregate step will be as follows:

Standard Aggregate Σ

Columns

Input	Aggregation Mode	Output
Name: MEMBER_ID Type: StringType	<input checked="" type="radio"/> Group <input type="radio"/> f(Agg)	Name: MEMBER_ID Type: StringType
Name: PLUSES Type: IntegerType	<input type="radio"/> Group <input checked="" type="radio"/> f(Agg) sum	Name: PLUSES_SUM Type: IntegerType
Name: MINUSES Type: IntegerType	<input type="radio"/> Group <input checked="" type="radio"/> f(Agg) avg	Name: MINUSES_AVG Type: FloatType

Buttons: Refresh, Cancel, OK

We choose Group for the column(s) (in this case, MEMBER_ID) that we are aggregating on, and f(Agg) for the columns that we are aggregating (in this case, summing on PLUSES and averaging on MINUSES). The interface suggests output column names and types for the aggregation columns, which you can edit.

In this case, the output will be as follows:

MEMBER_ID	PLUSES_SUM	MINUSES_AVG
1006	4	3.0
1007	8	2.0
1003	13	5.75
1002	15	6.3333335

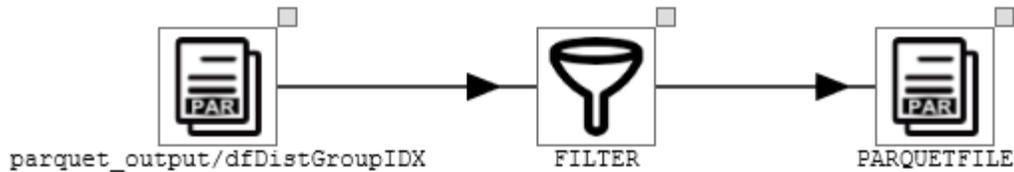
Filter

Introduction

The Filter step allows you to filter the data flow, using an expression that, in SQL, would be placed in a WHERE clause. Note that the interface does not parse the expression for syntax errors before runtime; if the expression is complex or if you are unsure of your SQL syntax, it is best to try it out in an SQL parser beforehand.

Example,

In this example, we use a Parquet file as input to the Filter step:



The input appears as follows:

The screenshot shows a console window with a 'Data Source' field containing '/parquet_output/dfDistGroupIDX'. Below the field is a table of data with columns 'group_id', 'user_id', and 'user_name'.

group_id	user_id	user_name
3	8	samuel
3	9	kingfisher
3	10	rakesh
2	4	harry
2	5	john
2	6	nandan
2	7	bush
1	1	zealot
1	2	salma
1	3	george

Inside the Filter step's properties, we click on Expression Builder, and enter an expression corresponding to a WHERE clause.

fx Expression ✕

EXPRESSION

Expression Attributes

COLUMN	<input type="text"/>	FUNCTION	<input type="text"/>
CONSTANT	<input type="text"/>	UDF	<input type="text"/>
VARIABLE	<input type="text"/>	SEQUENCE	<input type="text"/>
OPERATOR	<input type="text"/>		

Expression

```
'group_id'>0
```

Filter ✕

Input Columns  Filter Expression

group_id user_id user_name	<pre>'group_id'>0</pre>
----------------------------------	----------------------------

Join

Introduction

The Join step performs SQL-type joins on datasets.

Example,

Suppose we have two tables, MEMBER_DATA and MEMBER_SCORES, containing the data

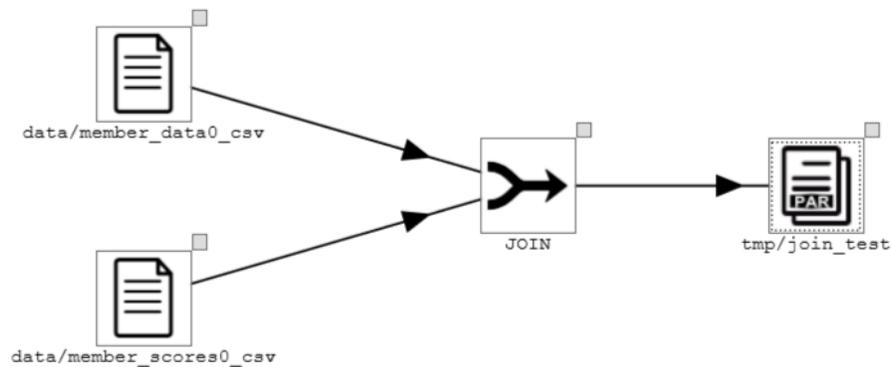
MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1005	eeee
1006	ffff

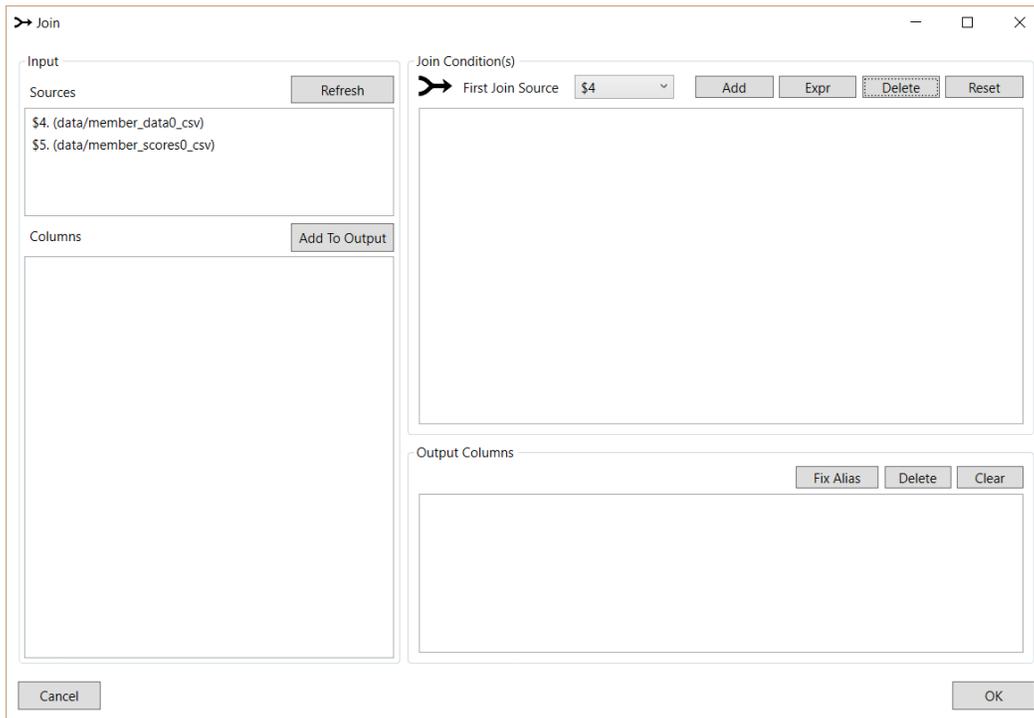
and

MEMBER_ID	PLUSES	MINUSES
1002	4	5
1003	6	6
1006	4	3
1007	8	2

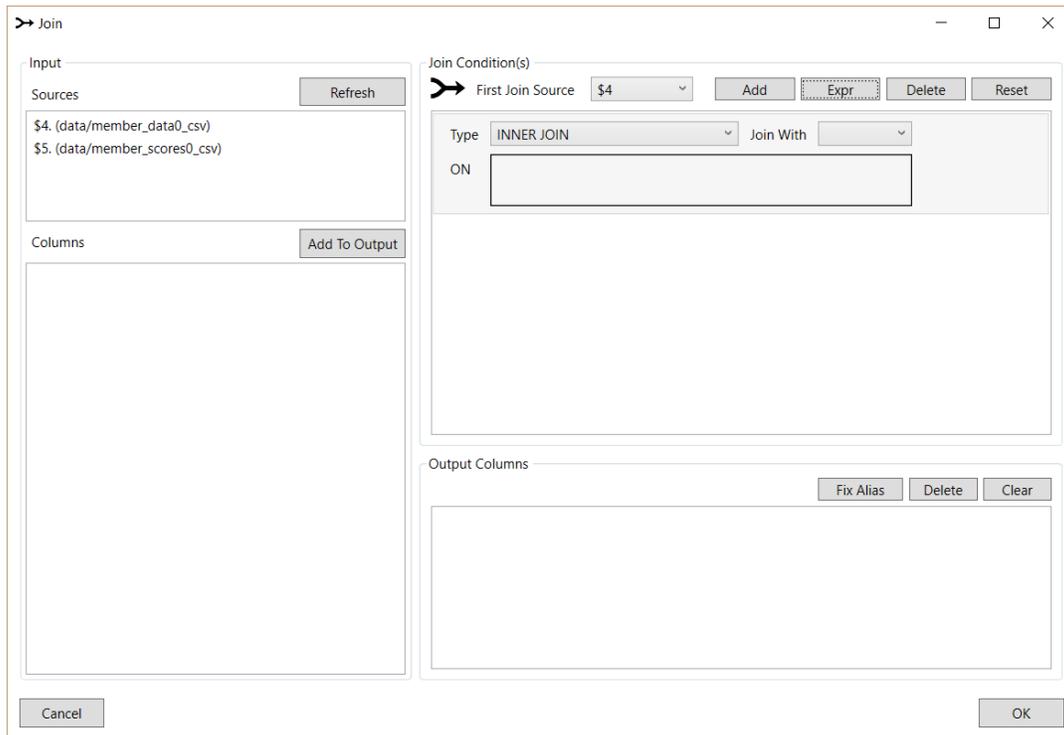
respectively.

We'll write a job to use the Join step to do an inner join on these two tables.



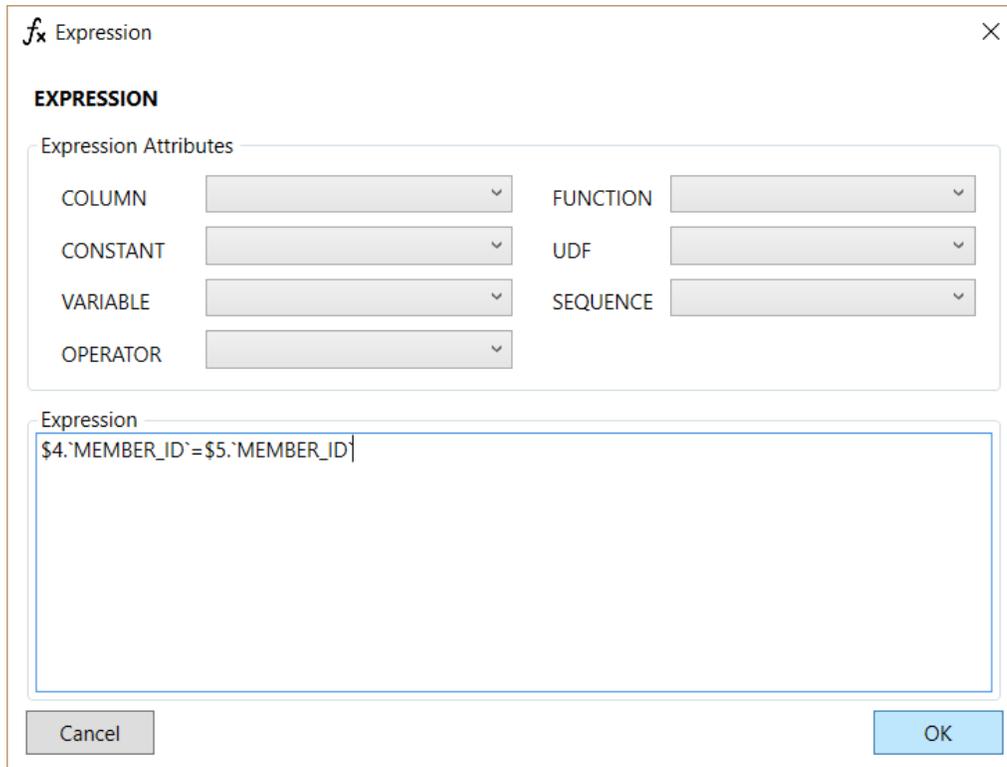


First, assign the role of First Join Source (or Left join source) to MEMBER_DATA, by choosing its number (in this case, \$4) from the drop-down list. Then Click on [Add] to add a join expression:



Choose the join type from the Type drop-down list. In this case, we'll keep the default choice, INNER JOIN. Choose the number for MEMBER_SCORES in the Join With drop-down list. (This seems superfluous in this case, since there is only one other table, but Join can take more than two inputs, so in general, there may be more than one choice.) Now click on the Expr button.

This will bring up an expression editor:

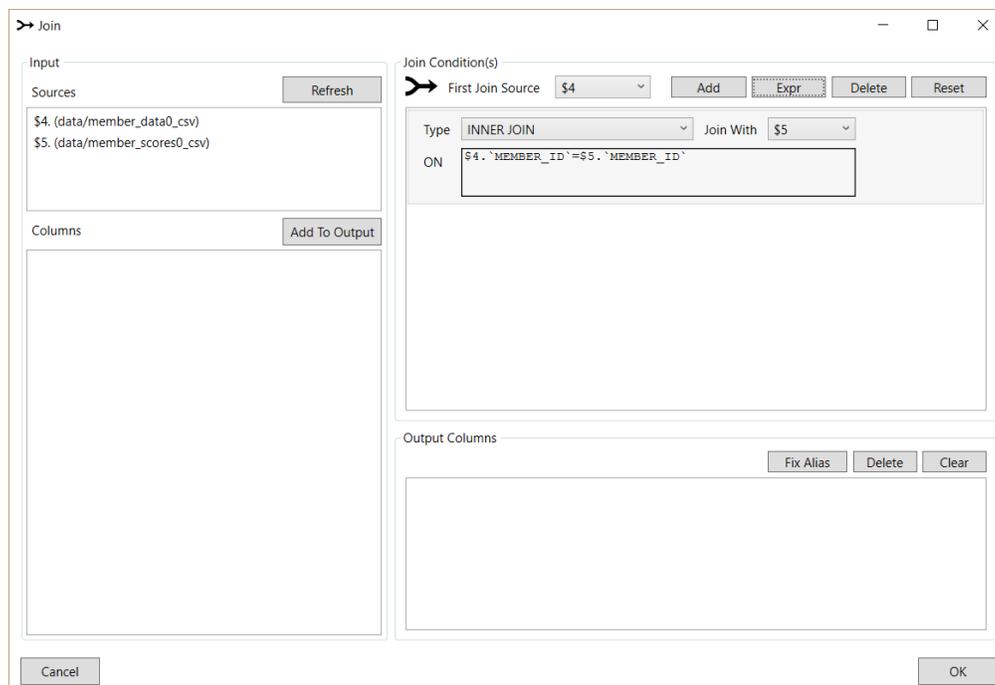


The dialog box is titled "Expression" and contains a section for "EXPRESSION" with "Expression Attributes". There are six dropdown menus: COLUMN, CONSTANT, VARIABLE, OPERATOR, FUNCTION, UDF, and SEQUENCE. Below these is a text area containing the expression `$4.`MEMBER_ID`=$5.`MEMBER_ID``. At the bottom are "Cancel" and "OK" buttons.

Enter an appropriate join expression in the editor – in our case, simply `$4.`MEMBER_ID`=$5.`MEMBER_ID``. (Remember that column names must be surrounded by backquotes (`), as shown in the example. The COLUMN dropdown list will automatically supply them, but if you type the column names in yourself, the backquotes are your responsibility!)

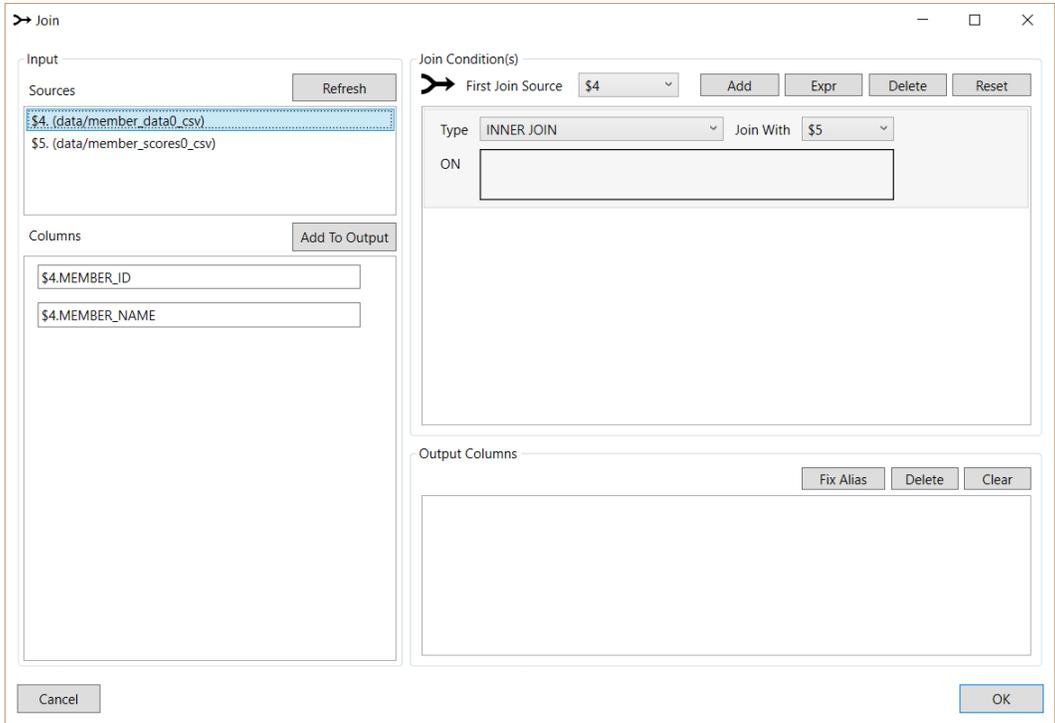
Click [OK] to return to the properties window.

Now we must decide what columns get mapped from the input to the output.

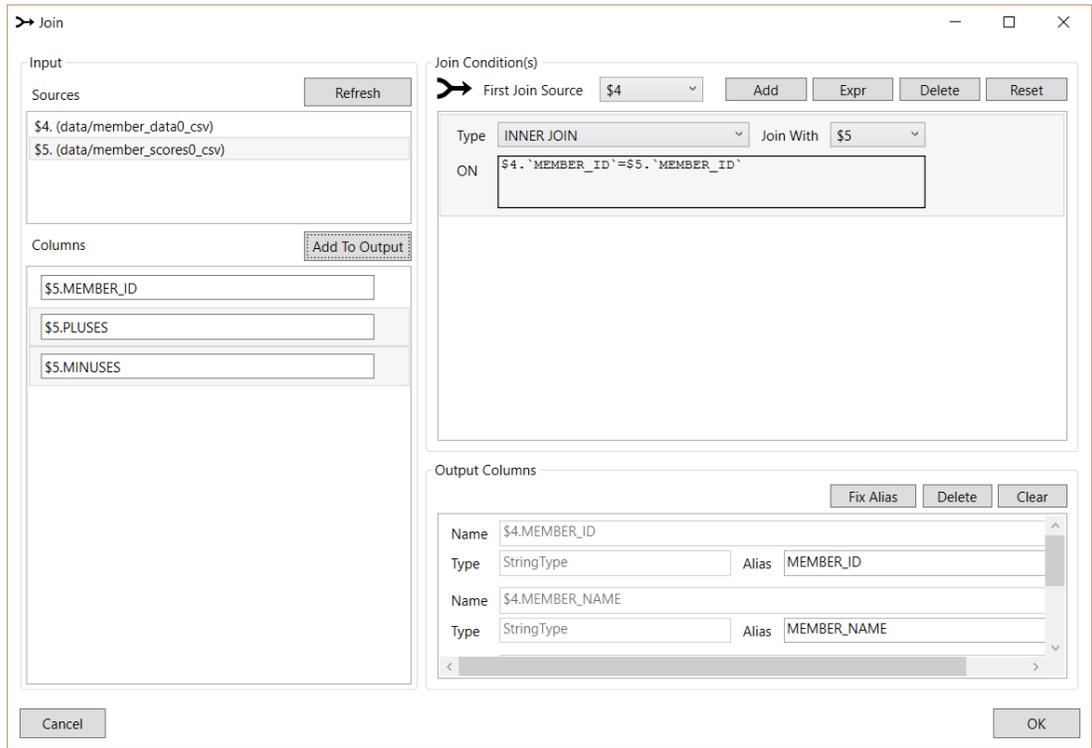


The dialog box is titled "Join" and is split into two main sections. The left section, "Input", has "Sources" with a "Refresh" button and a list containing `$4. (data/member_data0_csv)` and `$5. (data/member_scores0_csv)`. Below is a "Columns" section with an "Add To Output" button. The right section, "Join Condition(s)", has a "First Join Source" dropdown set to `$4` and buttons for "Add", "Expr", "Delete", and "Reset". Below this, "Type" is set to "INNER JOIN" and "Join With" is set to `$5`. The "ON" clause contains the expression `$4.`MEMBER_ID`=$5.`MEMBER_ID``. At the bottom right are "Fix Alias", "Delete", and "Clear" buttons. "Cancel" and "OK" buttons are at the bottom of the dialog.

The input tables are listed in the upper left-hand corner, under Sources. Clicking on each source will cause that source's columns to appear in the Columns section:



Then select the columns you wish to add to the output and Click the Add to Output button. You can select multiple columns at once by using the shift key. In our case, we'll add both MEMBER_ID and MEMBER_NAME from the MEMBER_NAMES dataset to the output. Then Click on the MEMBER_SCORES source and add PLUSES and MINUSES to the output.



Click [OK] to exit the properties window.

You will still need to complete the job by mapping the join output to an output stage and setting up the output stage by giving it the name of an output file and setting its other properties.

After running the job with the following input:

MEMBER_ID	MEMBER_NAME
1001	aaaa
1002	bbbb
1003	cccc
1004	dddd
1005	eeee
1006	ffff

and

MEMBER_ID	PLUSES	MINUSES
1002	4	5
1003	6	6
1006	4	3
1007	8	2

the output will be

MEMBER_ID	MEMBER_NAME	PLUSES	MINUSES
1002	bbbb	4	5
1003	cccc	6	6
1006	ffff	4	3

Had we chosen Join type of LEFT OUTER JOIN, the output would be

MEMBER_ID	MEMBER_NAME	PLUSES	MINUSES
1001	aaaa	null	null
1002	bbbb	4	5
1003	cccc	6	6
1004	dddd	null	null
1005	eeee	null	null
1006	ffff	4	3

The output for FULL OUTER JOIN would be

MEMBER_ID	MEMBER_NAME	PLUSES	MINUSES
1006	ffff	4	3
1003	cccc	6	6
1002	bbbb	4	5
1004	dddd	null	null
1005	eeee	null	null
1001	aaaa	null	null
null	null	8	2

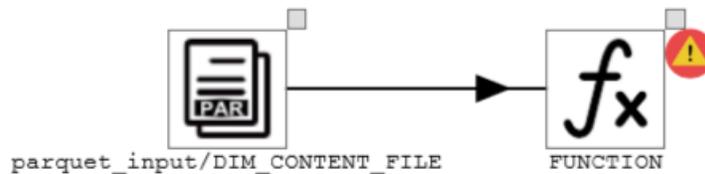
Function

Introduction

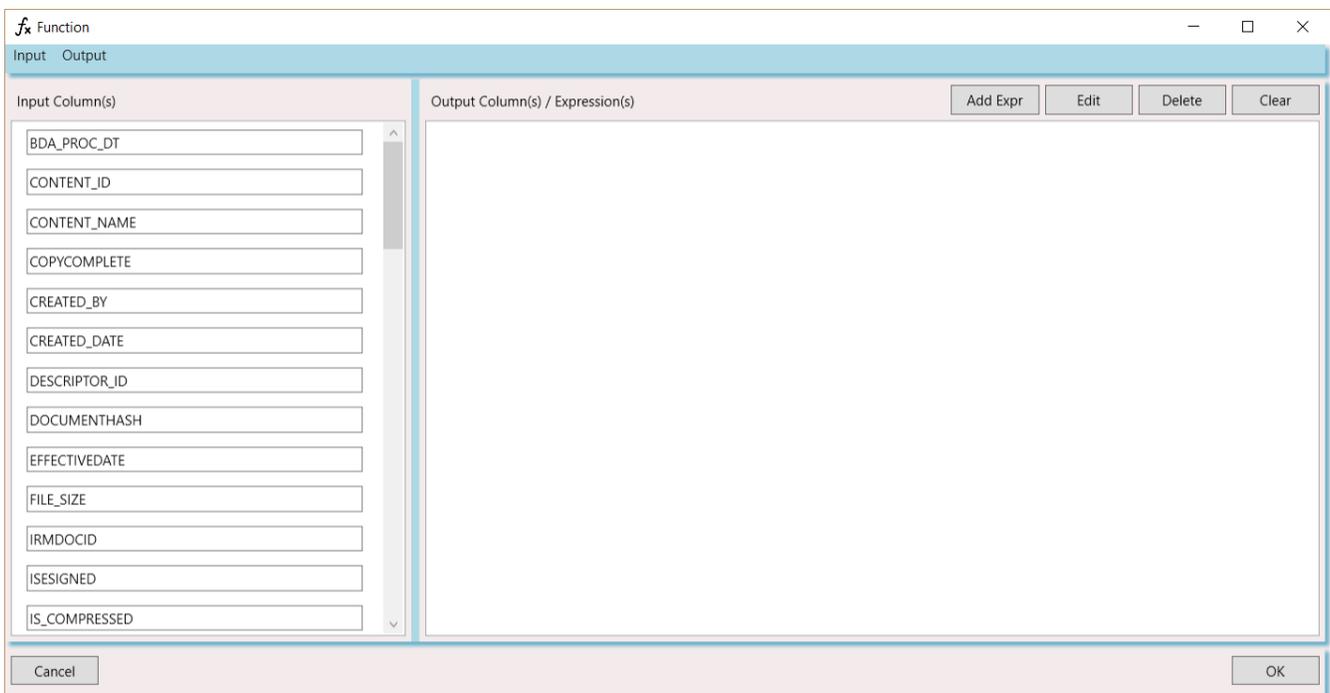
The Function step allows you to apply functions to columns and combinations of columns. The functions may be mathematical functions, string manipulation functions, date and time manipulation functions, conversion functions, or other functions appropriate to the data types. The Function step allows you to create new columns and to drop exiting columns.

Example,

The Function step must be connected to a source of metadata for you to access its properties. Suppose the Function step is connected to a data source as shown below:



The properties window for the Function step will initially appear as follows:



fx Expression

EXPRESSION

Expression Attributes

COLUMN FUNCTION

CONSTANT UDF

VARIABLE SEQUENCE

OPERATOR

Expression

UPPER(MEMBER_NAME)

Cancel OK

Project Column(s)

Structure (Expression-Alias-DataType)

fx Add Delete Edit

MEMBER_ID	MEMBER_ID	INTEGER
UPPER(MEMBER_NAME)	MEMBER_NAME	VARCHAR(30)
SQRT(MEMBER_ID)	REAL_MEMBER_ID	DOUBLE

Cancel OK

Original column left unchanged

Column altered by function

New column created from old column

Input and output for this example:

MEMBER_ID	MEMBER_NAME
1004	"dddd"
1002	"bbbb"
1001	"aaaa"
1003	"cccc"

MEMBER_ID	MEMBER_NAME	REAL_MEMBER_ID
1001	"AAAA"	31.63858403911275
1002	"BBBB"	31.654383582688826
1003	"CCCC"	31.670175244226233
1004	"DDDD"	31.68595903550972

SCD2

Introduction

SCD2 implements “Slowly Changing Dimensions Type 2.”

In data warehousing, some attributes of objects, such as, say, the prices of Veggie Burgers, may be referred to as *dimensions*, and stored as columns in *dimension tables*. Sometimes those attributes change, slowly relative to the frequency of loading the data warehouse – so, for example, a restaurant might raise the price of its Veggie Burgers. In this case they are referred to as *slowly changing dimensions*. It is often considered desirable to have a system that keeps track of the changes to dimensions, and makes it clear not only what the current value of the dimension is (in our example, what the current price of Veggie Burgers is), but also makes it possible to discover what the previous values were.

SCD2 is one of several ways to implement such a system. In SCD2, rows associated with dimension values are marked with effective timestamps and expiration timestamps. The row with the current dimension value will have the most recent effective timestamp and an expiration timestamp of “infinity” (usually something like 12/31/9999 23:59).

Example,

id	name	type_key	price	modified_date	dim_id	eff_ts	exp_ts
1	Chicken Burger	1	\$3.70	10/7/2016 19:03	1	10/7/2016 19:03	12/31/9999 23:59
2	Veggie Burger	1	\$3.20	10/7/2016 19:03	2	10/7/2016 19:03	12/31/9999 23:59
3	French Fries	2	\$2.00	10/7/2016 19:03	3	10/7/2016 19:03	12/31/9999 23:59
4	Twister Fries	2	\$2.20	10/7/2016 19:03	4	10/7/2016 19:03	12/31/9999 23:59

Consider the example shown above. Here we see the initial state of a price dimension table for restaurant food items. The fields *id* and *name* identify a certain food item, so, for example, *id* = 1 will always be a Chicken Burger, and *id* = 3 will always be French Fries. The *type_key* field identifies food type – for example, burgers, vs. fries. The *dim_id* is a unique identifier for the row. The *modified_date* field is the time when the values in the row were last changed, and will be the same as the effective timestamp. The effective timestamp (“*eff_ts*”) of all of the items is 10/7/2016 19:03, and the expiration timestamp (“*exp_ts*”) for all items is 12/31/9999 23:59, meaning that all items are current. In particular, the current price of a Veggie Burger is \$3.20.

Next, we update the price of Veggie Burgers, so that the price is now \$3.25.

id	name	type_key	price	modified_date	dim_id	eff_ts	exp_ts
1	Chicken Burger	1	\$3.70	10/7/2016 19:03	1	10/7/2016 19:03	12/31/9999 23:59
2	Veggie Burger	1	\$3.25	10/7/2016 20:08	5	10/7/2016 20:08	12/31/9999 23:59
2	Veggie Burger	1	\$3.20	10/7/2016 19:03	2	10/7/2016 19:03	10/7/2016 20:08
3	French Fries	2	\$2.00	10/7/2016 19:03	3	10/7/2016 19:03	12/31/9999 23:59
4	Twister Fries	2	\$2.20	10/7/2016 19:03	4	10/7/2016 19:03	12/31/9999 23:59

The table above has been updated to reflect the price change according to the SCD2 scheme. The row with the old price, identified by *dim_id* = 2, has been “expired,” by setting *exp_ts* to 10/7/2016 20:08. We have created a new row, identified by *dim_id* = 5, where price is now \$3.25. We know that the latter row, where price = \$3.25 and *dim_id* = 5 is the current row because its *exp_ts* is 12/31/9999 23:59. The table thus contains an indication of the current row as well a history of the previous prices.

How to Use the SCD2 Step

The SCD2 Step automates the maintenance of an SCD2 scheme like the one shown above. The user must provide appropriate input and dimension tables, and must label the columns as to their role. In the example below, we’ll show how to use the SCD2 step to obtain a dimension table like the one above.

The source for our dimension table is a table called **prices**, which initially looks like this:

id	name	type_key	price	modified_date
1	Chicken Burger	1	\$3.70	10/7/2016 19:03
2	VeggieBurger	1	\$3.20	10/7/2016 19:03
3	French Fries	2	\$2.00	10/7/2016 19:03
4	Twister Fries	2	\$2.20	10/7/2016 19:03

We must also create another table called **prices_dim**, consisting of all of the columns of the **prices** table plus, if they do not already exist, the columns **dim_id**, **eff_dt**, and **exp_dt**, which are BIGINT, TIMESTAMP, and TIMESTAMP respectively.

We will set up a simple, two-step ELT Maestro job as shown below:



The table step on the left is connected to the **prices** table. Now open the properties of the SCD2 step:

The screenshot shows the configuration window for a Slowly Changing Dimension Type II table. The window title is 'Slowly Changing Dimension Type II'. It has two tabs: 'General' and 'Dimension Property', with 'Dimension Property' currently selected. Under the 'Target SCD2 Dimension Table' section, there are three text input fields for 'Database', 'Schema', and 'Table', followed by a 'Browse' button. To the right of these fields is a small cube icon with 'SCD' and '2' on its faces. Below this section is a large empty text area labeled 'Columns (M=Match Rows, C=Check For Change, X=Ignore)'. At the bottom of the window are 'Cancel' and 'OK' buttons.

In the General tab, click on the Browse button and navigate to and select the **prices_dim** table:

Slowly Changing Dimension Type II

General Dimension Property

Target SCD2 Dimension Table

Database: 

Schema:

Table:

Columns (M=Match Rows, C=Check For Change, X=Ignore)

id	<input checked="" type="radio"/> M <input type="radio"/> C <input type="radio"/> X
name	<input checked="" type="radio"/> M <input type="radio"/> C <input type="radio"/> X
type_key	<input type="radio"/> M <input type="radio"/> C <input checked="" type="radio"/> X
price	<input type="radio"/> M <input checked="" type="radio"/> C <input type="radio"/> X
modified_date	<input type="radio"/> M <input type="radio"/> C <input checked="" type="radio"/> X
dim_id	<input type="radio"/> M <input type="radio"/> C <input checked="" type="radio"/> X
eff_ts	<input type="radio"/> M <input type="radio"/> C <input checked="" type="radio"/> X
exp_ts	<input type="radio"/> M <input type="radio"/> C <input checked="" type="radio"/> X

Each column in the dimension table needs to be classified as either a *match (M)*, *check for change (C)* or *ignore (X)* column. The columns that *identify* the object as the one described by the attribute (aka dimension) in question should be marked with an M. In our case, these columns are id and name. The dimension itself – the column or columns whose change we are tracking are marked with a C – in our case, the price column. All other columns should be marked with an X.

Slowly Changing Dimension Type II

General Dimension Property

Columns

- id
- name
- type_key
- price
- modified_date
- dim_id
- eff_ts
- exp_ts

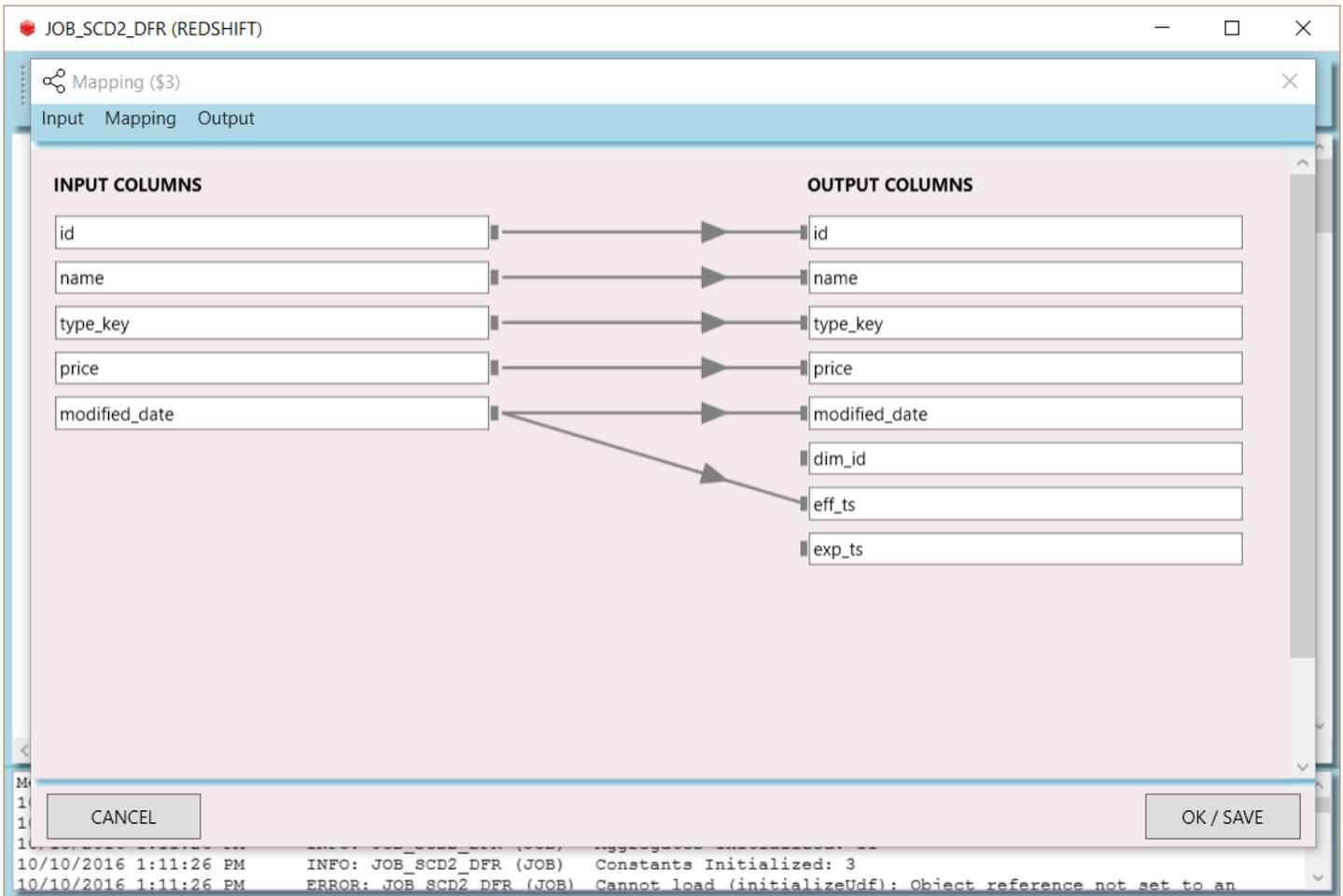
Column Roles (Drag and drop here)

DIM:

Effective_TS:

Expiration_TS:

On the Dimension Property tab, we specify which columns fill the roles of dimension identifier, effective timestamp, and expiration timestamp; in our case these will be the columns we created for this purpose. In some cases, a preexisting column from the input table may fill one of these roles – for example, we could have used modified date as our effective timestamp.



After clicking OK in the Dimension Property tab, the user is shown the mapping editor. In our case, we want to make sure that `modified_date` in the `prices` table maps to `eff_ts` (effective timestamp) as well as `modified_date` in the `prices_dim` table.

After clicking OK/SAVE, the job will be set up correctly to update the `prices_dim` table according to the SCD2 scheme.

Smart Script

Introduction

Smart script step is an item list processor. Let us define a simple use case to better understand how this step can be utilized.

Use Case

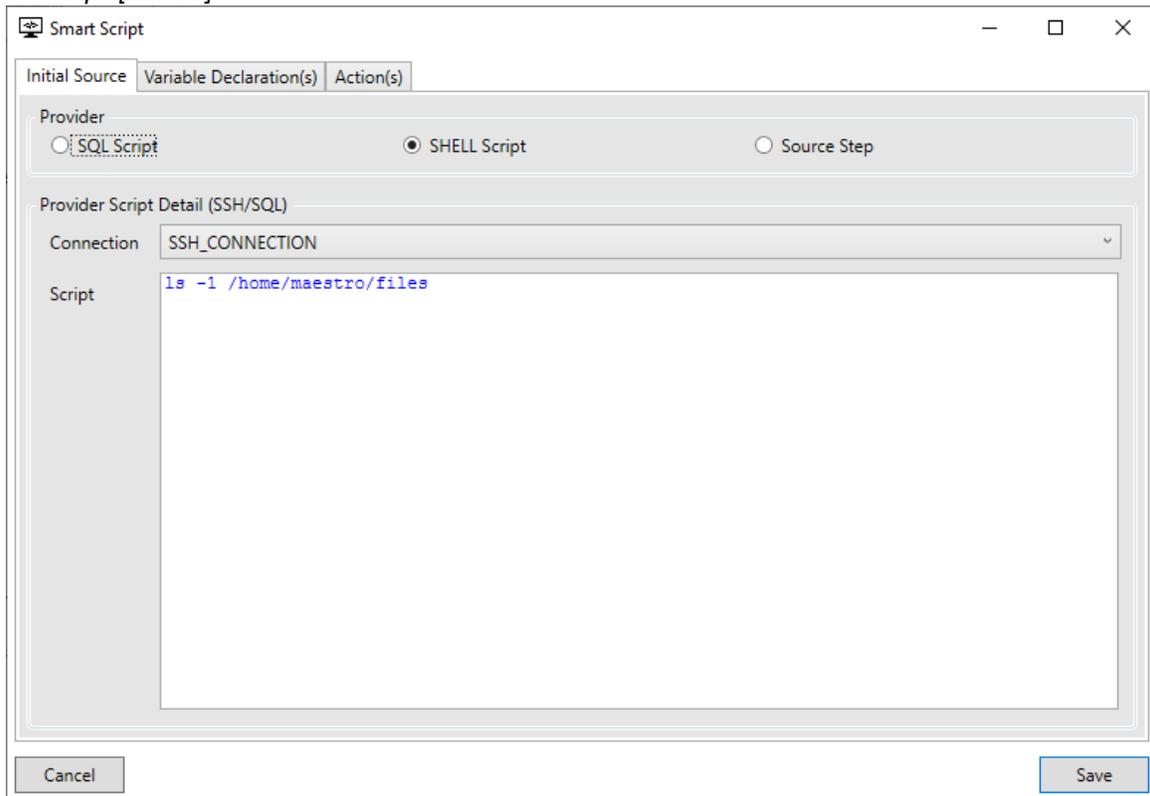
I have hundreds of text files on a Linux machine located at folder path(/home/maestro/files). I want to compress these files using gzip command. Ideally, I want to call 8 parallel gzip commands so that the process can be faster utilizing a multi core system. Then I want to run another script in parallel using 4 threads that does something with compressed files.

Design Template

- **[INPUT] is provided in form of a list of values generated by script.** This list can be output of a shell script or SQL. If shell script is provided the program splits standard output by new line and converts into array of values so that each line is a value item. If SQL script is provided the program retrieves array of value from the first column so that each row of first column is a value item.
- **[DECLARATIONS] are done to convert value from one form to another.** You can only use echo command to convert single input value to another.
- **[ACTIONS] are final execution of either SQL or SHELL script.** Variable declarations and conversions performed on previous step are referenced here. You can specify number of parallel calls in action.

Note: You can only declare and reference variables with \$ prefix.

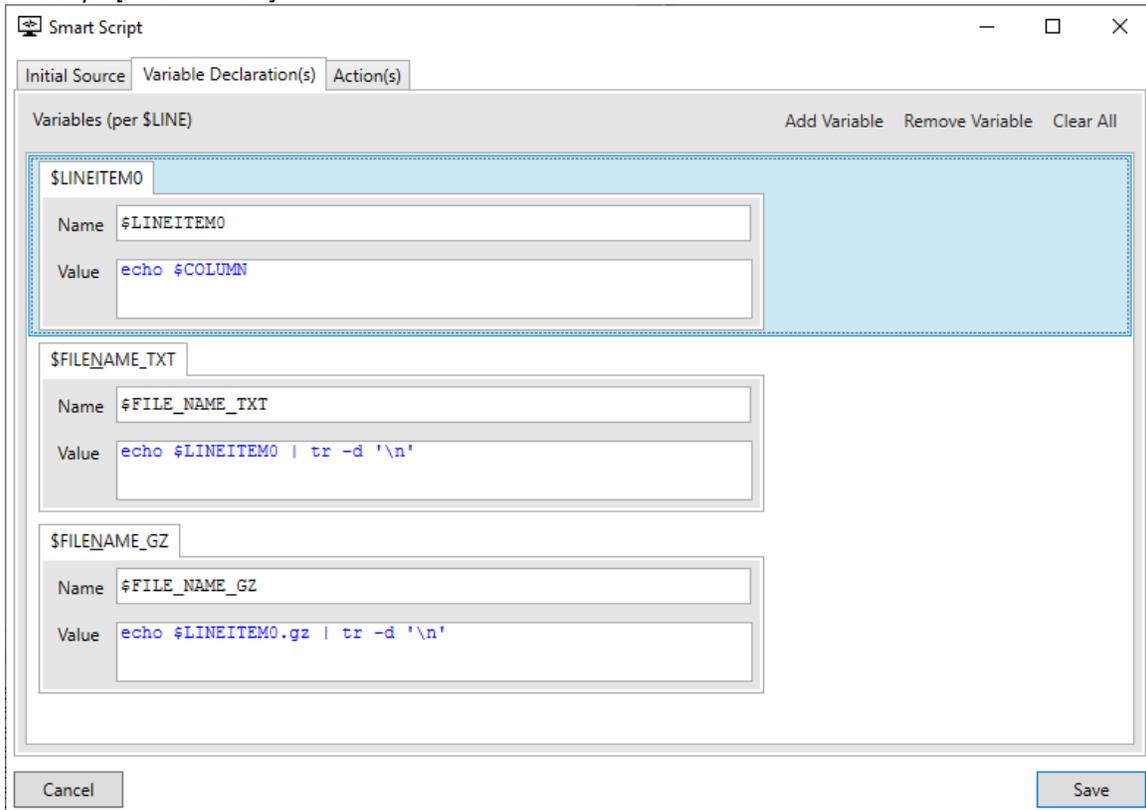
Figure: Smart Script [INPUT]



In this phase [INPUT]

I am running a shell script to list all file names located on a specified directory running ls command. My input command to generate files list is `ls -l /home/maestro/files`

Figure: Smart Script [Declaration]



In this phase [Declaration]

The first declaration `$LINEITEM0` has default value `$COLUMN`. Each output line from previous initial step is assigned to `$COLUMN` during runtime. The first declaration just copies that value into `$LINEITEM0` using `echo` command.

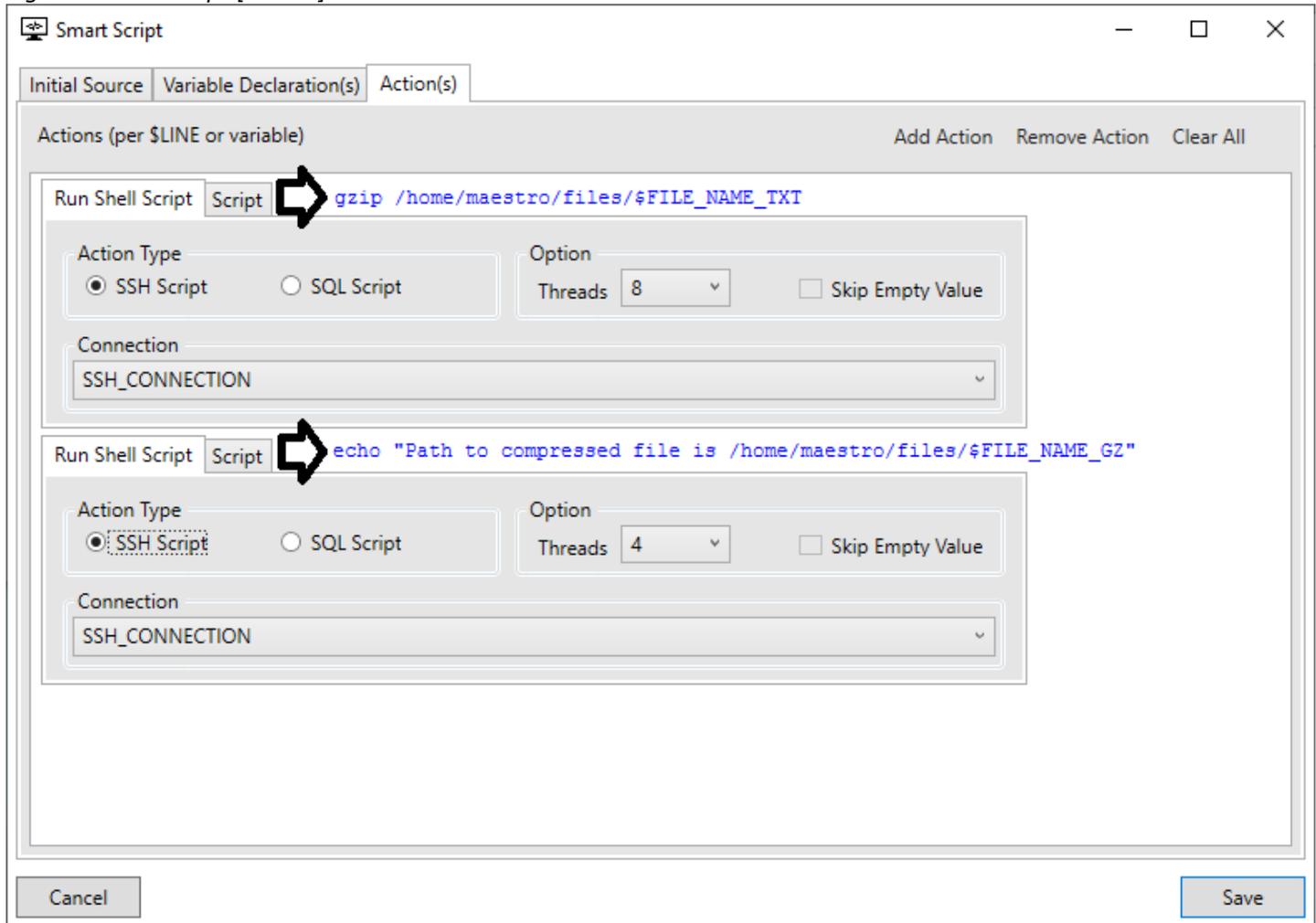
The second declaration `$FILE_NAME_TXT = echo $LINEITEM0 | tr -d '\n'` does pretty much the same thing, but it removes new lines from value and assigns to `$FILE_NAME_TXT`.

The third declaration `$FILE_NAME_GZ = echo $LINEITEM0.gz | tr -d '\n'` calculates that the compressed file name is going to be.

Why 3 declarations?

When I run `gzip` command, the name of file gets changed to it's original name and extra extension is added. So if input has value `file.186.txt`, the file name changes to `file.186.txt.gz` after running `gzip` command on action tab.

Figure: Smart Script [Action]



In this phase [Action]

I am running gzip command in parallel mode using 8 threads at a time until all files are compressed. Then I have second action which runs echo command using 4 threads at a time.

Rules

- Initial source must be provided by either a shell script or sql script
- Declarations can only be done by shell script
- Actions could be either

Support

Email: zdave@maestro-analytics.com

Website: www.maestro-analytics.com

