ETL design training session – basic

This document serves as a training session for using Talend Open Studio tool (from here on, we refer to it as Talend) for designing an ETL process that answers user analytical needs (expressed in a form of textual requirements).

Content:

Part A: Data vs. Control Flow In the first part of this training, we will briefly explain the difference between data flow (i.e., subjob) and control flow (i.e., job) in Talend and how to create them.

Part B: Create an ETL data flow (transformation) that answers the KPI During the second part, we will use an example textual requirement (KPI 1) to walk you through the process of creating an ETL data flow design using Talend.

Part C: Create control flow (job) to execute an ETL process Lastly, in the third part, we will create a control flow for managing the execution of our ETL process.

Lastly, in Appendix A, we will export the created Talend project into a .zip archive.

Data sources used in the training: All examples are created over the ACME Flying Use Case. For better understanding of internals of the system under the study, the schematic/diagrammatic representations of the domain and the available data sources are available (see slides: **FlyingUseCase.pdf**):

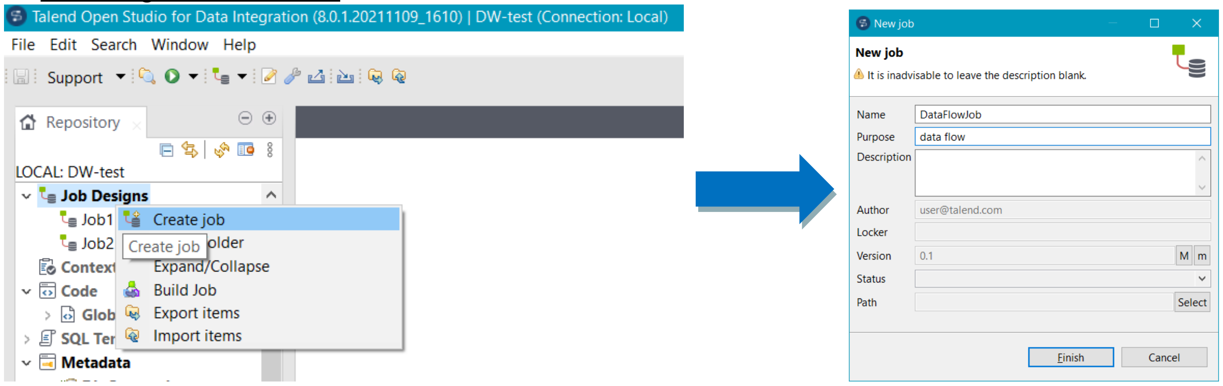
1) Diagrammatic/schematic representation of the **ACME subsystems** (AIMS and AMOS)
2) AIMS and AMOS DB schema (IE notation)
   a. With SQL to create **AIMS** and **AMOS tables (AIMS.sql and AMOS.sql)**.
3) External sources:
   a. aircraft-manufaturerinfo-lookup.csv
   b. maintenance-personnel-airport-lookup.csv

ETL design: training session - basic

Part A: Data vs. Control Flow

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

- <u>**Data flow**</u> is in charge of performing various operations over data themselves in order to prepare them for loading into a DW (or directly for exploiting them by end users). That is, data **extraction** (reading from the data sources), various data **transformation** tasks (data cleaning, integration, format conversions, etc.) and finally **loading** of the data to previously created target data stores of the DW. Data flows are typically executed as **pipelines** of operations (rather than a set of strictly sequential steps started one after another), meaning that they all start executing at the same time but keep idle until they receive data at the input. In Talend, data flows are called **jobs (or subjobs if they belong to a control flow job)**, and they are created as follows: <u>**Right click on Job Designs-> Create Job**</u>



New job

It is inadvisable to leave the description blank.

Name	DataFlowJob
Purpose	data flow
Description	
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	

- **Control flow**, on the other side is responsible of orchestrating the execution of one or more data flows. It does not work directly with data, but rather on managing the execution of data processing (i.e., scheduling, starting, checking for possible errors occurred during the execution, etc.). Unlike data flow, in control flow the order of execution is strictly defined by the sequence of activities, meaning that one activity does not start its execution until all its input activities have finished. This is especially important in the case of dependent data processing, where the results of one data flow are needed before starting the execution of another. In Talend, control flows are also called **jobs**, but they consist of several subjobs each one representing one data flow. Control flows are created within the same environment as data flows, but by using the **tRunJob**[1] component from the **Orchestration** palette. Each of these components references some previously created subjob (data flow) and in combination with other components from the **Orchestration** palette[2] creates a control flow job.

Part B: Create an ETL data flow (transformation) that answers the KPI

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

<u>**KPI 1**</u>

"Analyze the *total number of* ***flight hours*** *(actualArrival – actualDeparture) of an aircraft* from the point of view of the ***aircraft's model*** *(aircraft_model) and* ***year*** *(extract(year from actualDeparture)) of the flight with* ***number of passengers*** *(passengers)* greater than 50. "

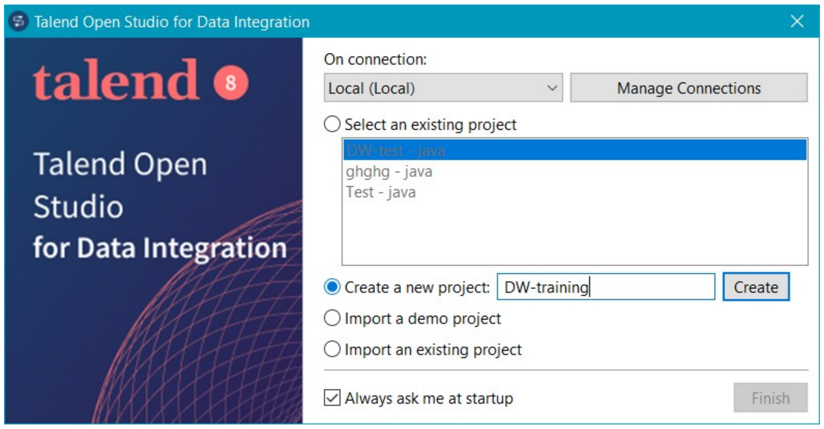
[1] https://help.talend.com/r/en-US/8.0/orchestration/trunjob
[2] https://help.talend.com/r/en-US/8.0/orchestration/orchestration

<u>Design process:</u>

The ETL design in Talend should be provided using their graphical editor (**Talend Open Studio for Data Integration**).

1. **Opening Talend's graphical ETL editor and creating new Talend project**

After we open Talend, we first see the following screen


Talend Open Studio for Data Integration
talend 8
Talend Open Studio for Data Integration
Manage Connections
Create
Finish

On connection: Local (Local)

Select an existing project: DW-test - java, ghghg - java, Test - java

Create a new project: DW-training

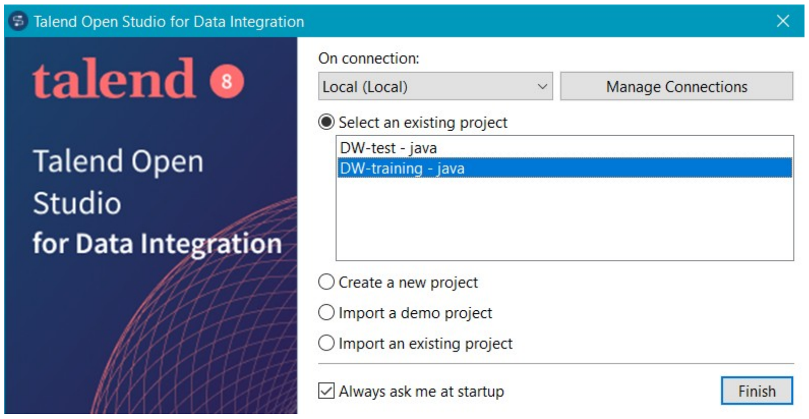
Import a demo project

Import an existing project

Always ask me at startup

Since we are working in local (without remote ETL server) we will leave the option Local for the connection. If it's the first time we open Talend (or we want to start a new project) we should choose the option **Create a new project**, give it a name (e.g., DW-training) and press the button **Create**.

After we create a new project, it will appear in the list of existing projects (as seen in the figure below). You should select it and press **Finish** to start the tool.


Talend Open Studio for Data Integration
talend 8
Talend Open Studio for Data Integration
Manage Connections
Finish

On connection: Local (Local)

Select an existing project: DW-test - java, DW-training - java

Create a new project

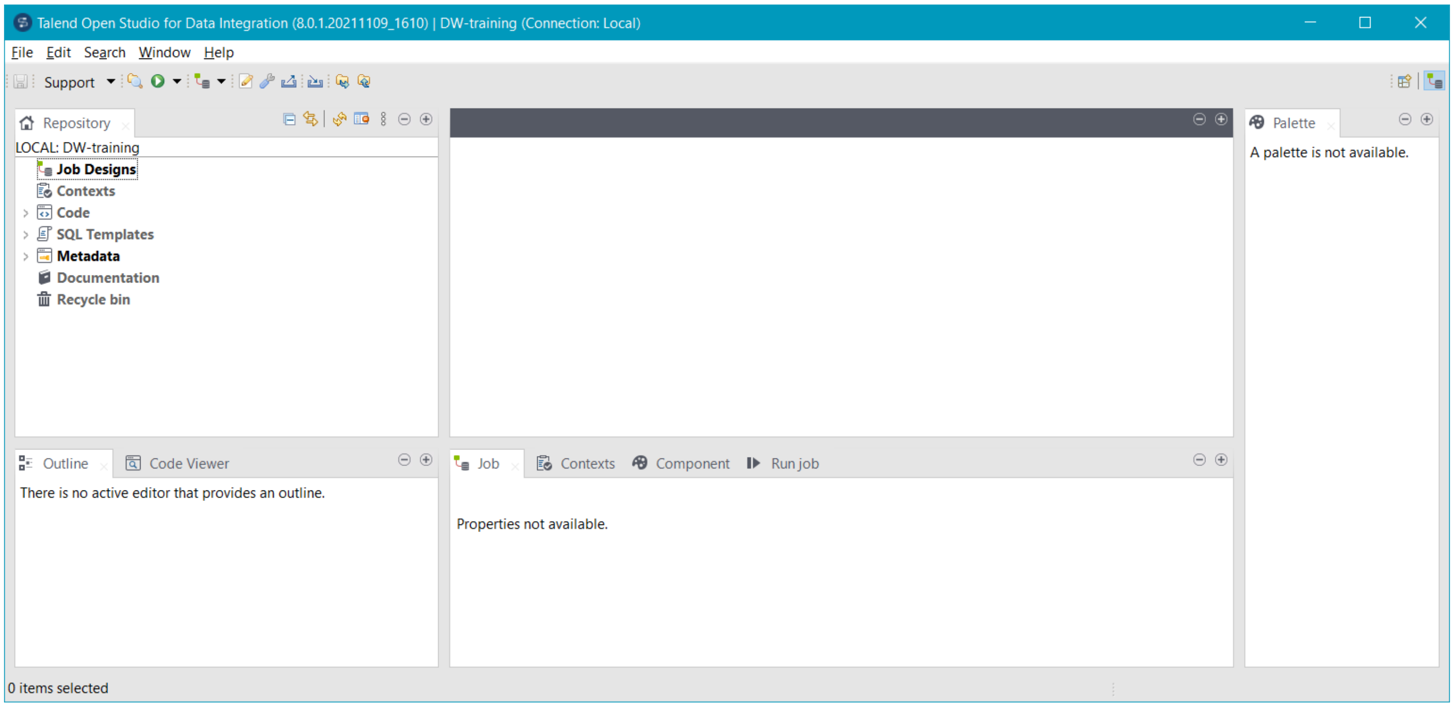
Import a demo project

Import an existing project

Always ask me at startup

Notice that in the future, instead of creating a new project, you should choose the option **Select an existing project**, select the project you want to open and click **Finish**.

After we open a new or existing project in Talend, we are supposed to see the following starting screen (please notice that depending on the version you are running, the welcome screen can differ):



Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW-training (Connection: Local)

A palette is not available.

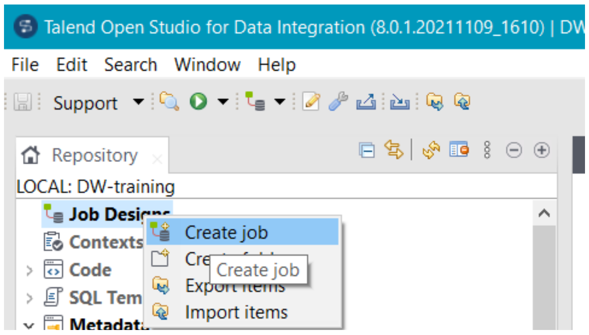
There is no active editor that provides an outline.

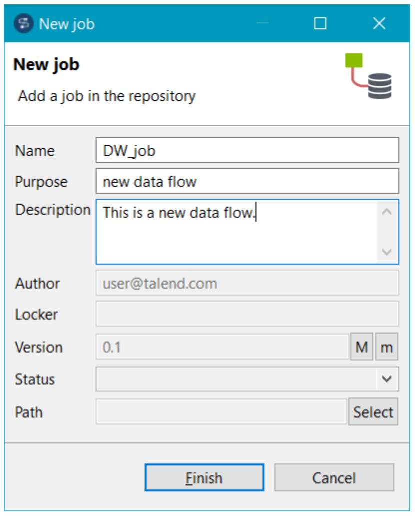
Properties not available.

0 items selected

2. Creating empty data flow (job):

To create new data flow (for designing an ETL flow) we right click on **Job Designs** inside the **Repository** pane and choose the **Create job** option, as shown in the figure below-left. We should then provide at least a name for the newly created job and press **Finish**, as shown in the figure below-right.





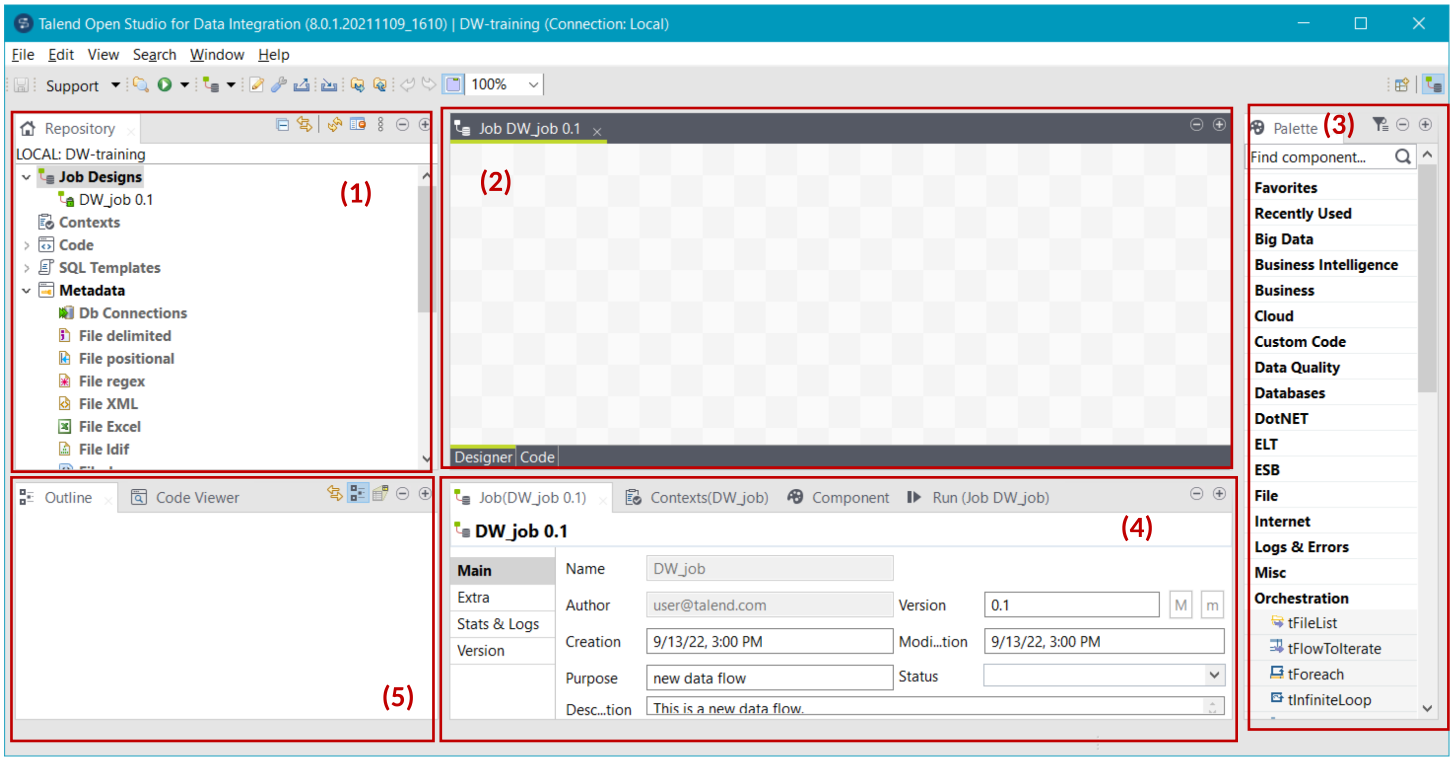
New job

New job

Add a job in the repository

Name	DW_job
Purpose	new data flow
Description	This is a new data flow.
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	

After we create a new job, the empty design canvas should be visible in the screen as shown in the figure below.



Name	DW_job		
Author	user@talend.com	Version	0.1
Creation	9/13/22, 3:00 PM	Modi...tion	9/13/22, 3:00 PM
Purpose	new data flow	Status	
Desc...tion	This is a new data flow.		

We should distinguish 5 main panes of the Talend interface (see figure above)

(1) **Repository** of the project where we will find all the jobs and subjobs created inside the project (under **Job Designs**), as well as the project **Metadata**, from which we will specifically use **Db Connections**.

(2) Designer canvas of the currently selected job/subjob.

(3) **Palette** of **components** that we can include in our data or control flows.

(4) Control pane where we can find the **Job** tab for job and transformation configuration, as well as the **Run** tab for runtime configuration and running of the current job.

(5) **Outline** of the ETL components included in the flow and the **Code Viewer** where the designer can see the translation of each component into Java code. For this lab, we will not be using this panel.

3. Defining data source inputs of the ETL process:

First, we need to identify the sources from where the ETL process is supposed to read the data. Considering the Flying use case, four different sources are available **1)** AMOS DB **2)** AIMS DB, **3)** maintenance-personnel-airport-lookup.csv (Personnel with the airport where they are employed), and **4)** aircraft-manufaturerinfo-lookup.csv (Aircraft with their, model and manufacturer).

Considering our example, we identify the following attributes that need to be extracted from the identified data sources:

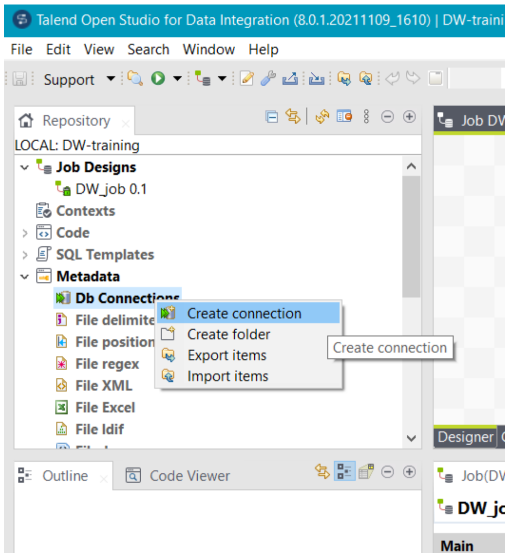
- ***Flight hours (actualArrival – actualDeparture):***
  - ***actualArrival*** *maps to the* ***Flights*** *table of* ***AIMS DB*** *and its column* ***actualArrival.***
  - ***actualDeparture*** *maps to the* ***Flights*** *table of* ***AIMS DB*** *and its column* ***actualDeparture.***
- ***Aircraft's model*** *maps to* ***the aircraft-manufaturerinfo-lookup.csv*** *file and its column* ***aircraft_model****.*
- ***Year*** *maps to the* ***Flights*** *table of AIMS DB and its column* ***actualDeparture*** *with the SQL function applied to extract year from the timestamp:* ***extract (year from actualDeparture) as year****.*
- ***Number of passengers*** *maps to the* ***Flights*** *table of* ***AIMS DB*** *and its column passengers.*

For each of these mappings (coming from a distinct data source) we need to define the data input component. Input component templates are provided in the **Palette** pane located at the right side of the Talend window.

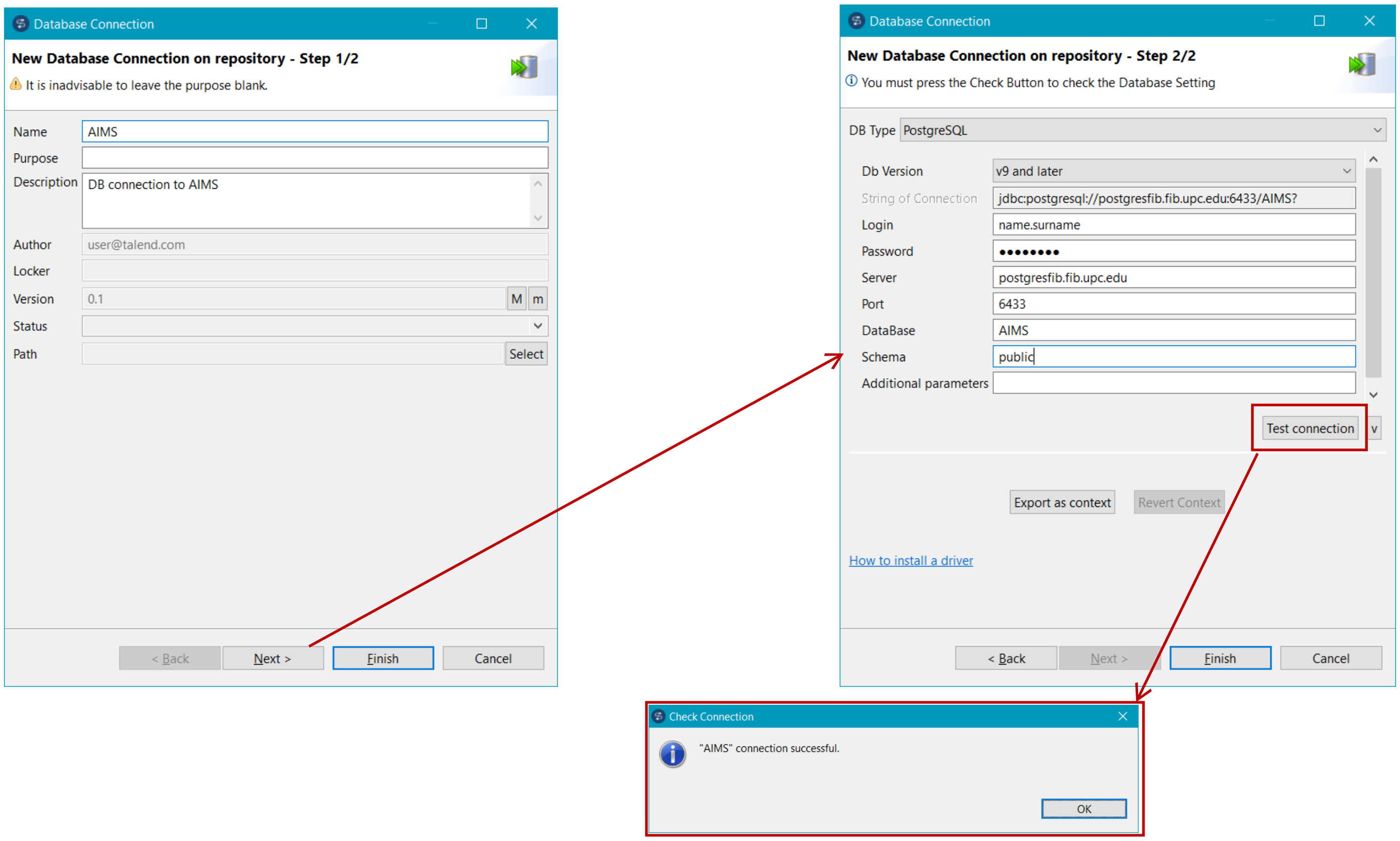
Defining DB connection metadata

Before we start adding data input components, we will first create a DB connection to the AIMS PostgreSQL DB, that we will use throughout this tutorial. In the similar manner, you should define other connections (e.g., with the AMOS database, or with the Oracle database for the output DW).

We go the **Metadata** section of the project **Repository**, right-click over the **Db Connections** option and choose **Create connection**, as shown in the figure below.



In the dialogs below, we need to give a name to the connection (e.g., AIMS), click **Next**, and then define the connection parameters as shown in the figure below (Importantly, you should use your UPC username and as a password use the provided password in the form **DBddmmyy** – where **dd** is day, **mm** is month, and **yy** is the year of your birth). Remeber that you must be previously connected to the UPC VPN.



Database Connection

New Database Connection on repository - Step 1/2

It is inadvisable to leave the purpose blank.

Name	AIMS
Purpose	
Description	DB connection to AIMS
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	

Database Connection

New Database Connection on repository - Step 2/2

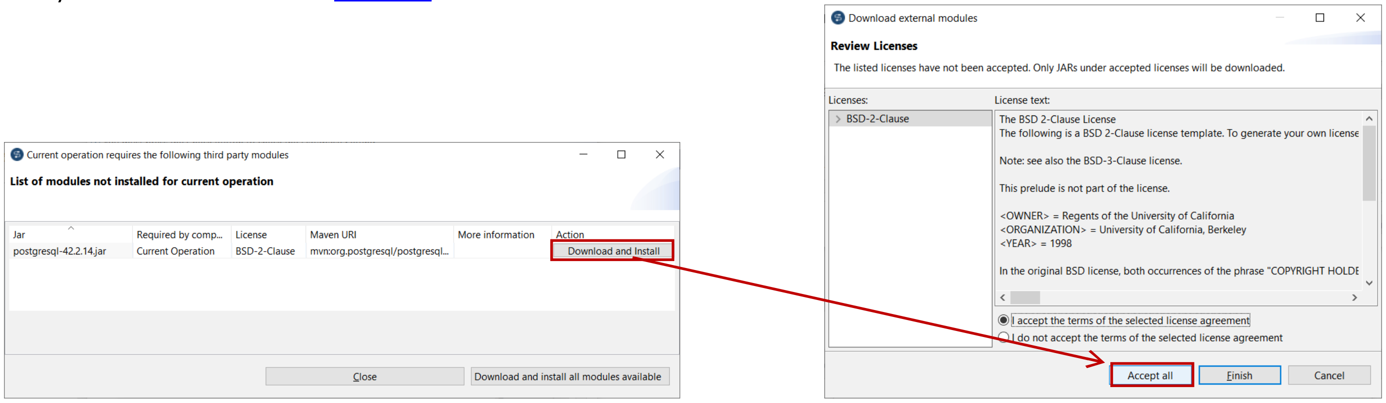
You must press the Check Button to check the Database Setting

DB Type	PostgreSQL
Db Version	v9 and later
String of Connection	jdbc:postgresql://postgresfib.fib.upc.edu:6433/AIMS?
Login	name.surname
Password	••••••••
Server	postgresfib.fib.upc.edu
Port	6433
DataBase	AIMS
Schema	public
Additional parameters	

Check Connection

"AIMS" connection successful.

After clicking **Test connection**, only the first time, Talend will prompt a window to install a JDBC driver for connecting to a specific DBMS, in this case PostgreSQL. In the list, it will appear postgres Jar and you should click on the **Download and Install** and then **Accept all**, as it is shown in the figure below. After the installation, Talend will return the message whether the connection is successful or not, as in the above figure. If not successful, please revise the connection parameters, and ensure that you are connected to the [UPC VPN](UPC VPN).


Current operation requires the following third party modules
Download and Install
Close
Download and install all modules available
Download external modules
Accept all
Finish
Cancel

List of modules not installed for current operation

Jar	Required by comp...	License	Maven URI	More information	Action
postgresql-42.2.14.jar	Current Operation	BSD-2-Clause	mvn:org.postgresql/postgresql...		Download and Install

Review Licenses

The listed licenses have not been accepted. Only JARs under accepted licenses will be downloaded.

Licenses: BSD-2-Clause

License text:

The BSD 2-Clause License
The following is a BSD 2-Clause license template. To generate your own license

Note: see also the BSD-3-Clause license.

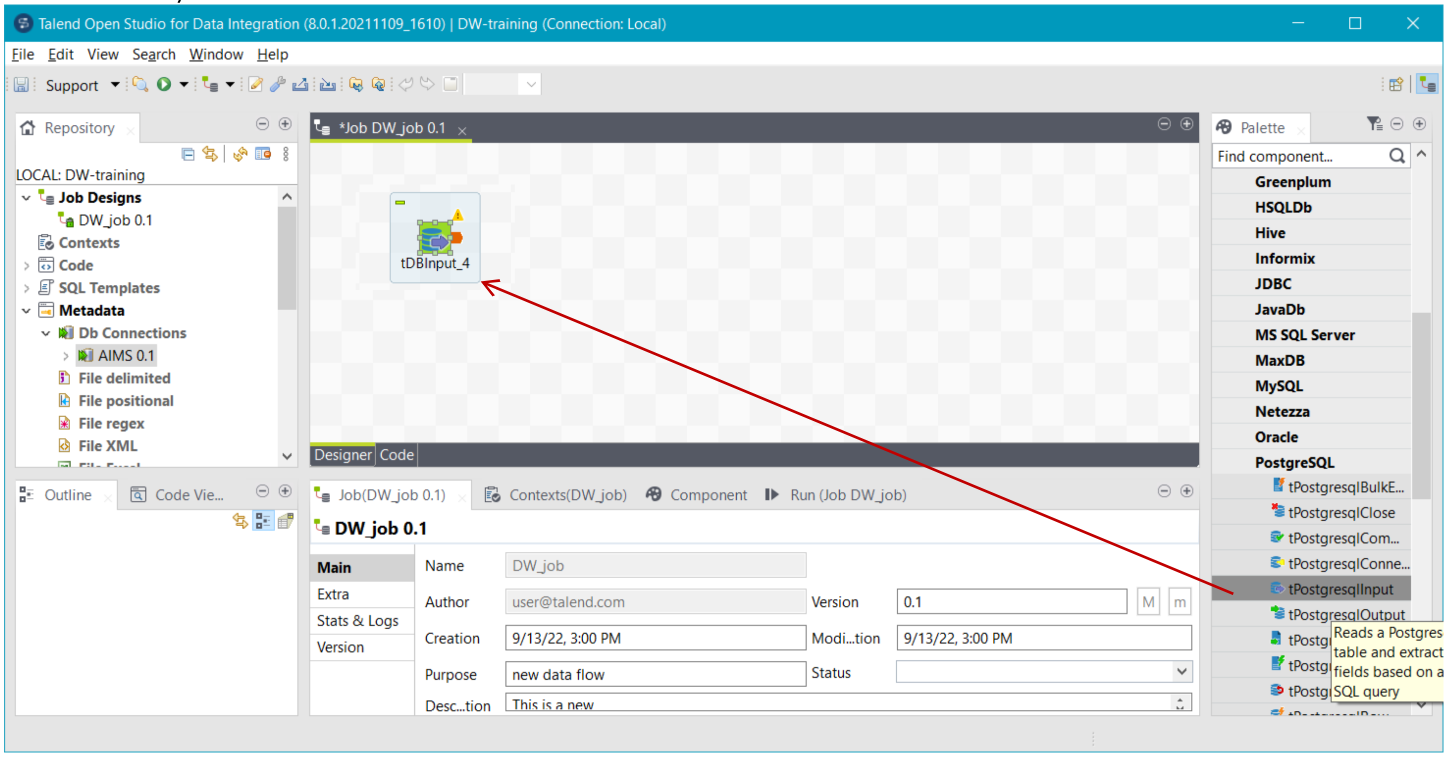
This prelude is not part of the license.

<OWNER> = Regents of the University of California
<ORGANIZATION> = University of California, Berkeley
<YEAR> = 1998

In the original BSD license, both occurrences of the phrase "COPYRIGHT HOLDE

(●) I accept the terms of the selected license agreement
( ) I do not accept the terms of the selected license agreement

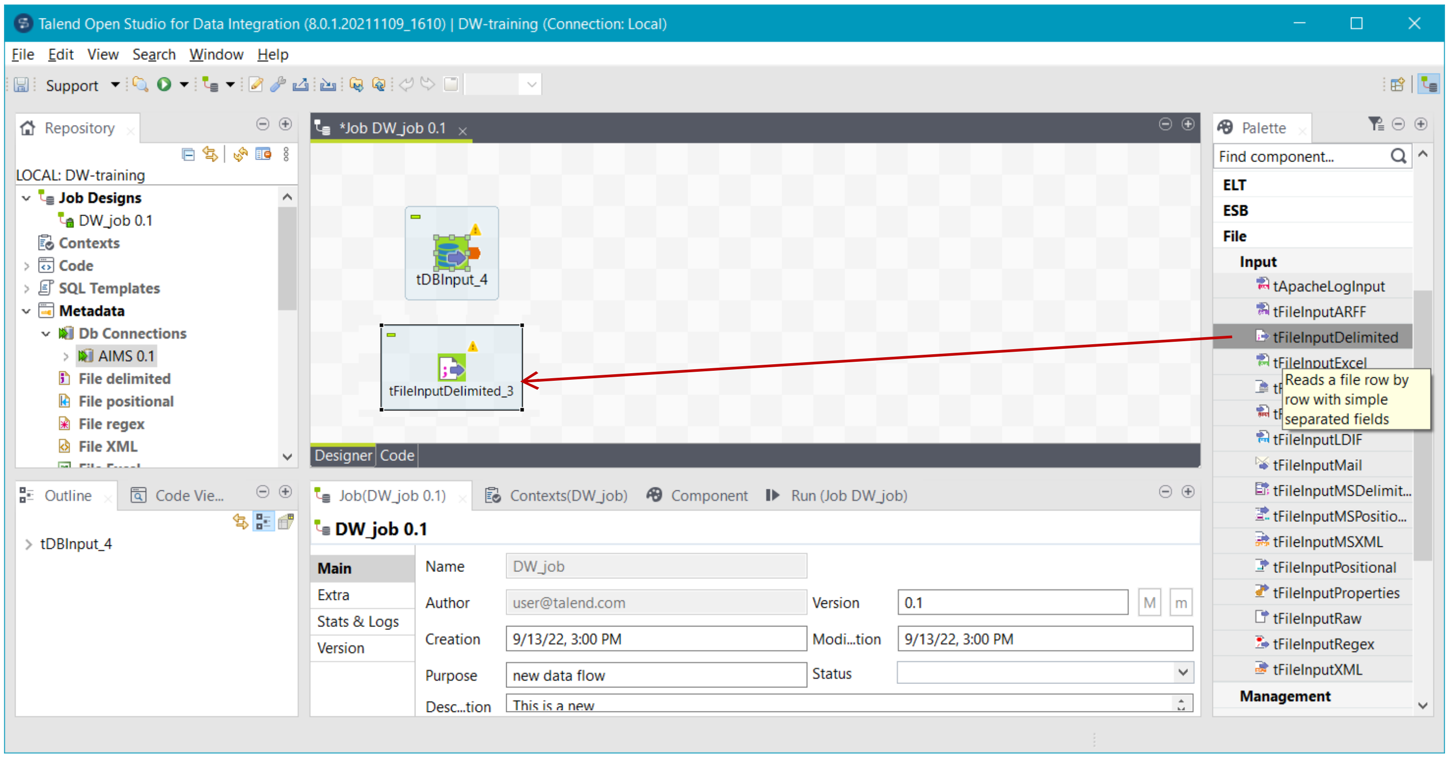
Once we have the DB connection created, we can proceed with adding a data input component to our Talend data flow. Inside the **Palette** tab, we choose the Databases->DB Specifics->PostgreSQL->**tPostgresqlInput** transformation (as shown in the figure below) and drag it to the designer canvas. Notice that you can always use the **Find component…** option to quickly locate the transformation you need.



Reads a Postgres table and extract fields based on a SQL query

Name	DW_job		
Author	user@talend.com	Version	0.1
Creation	9/13/22, 3:00 PM	Modi...tion	9/13/22, 3:00 PM
Purpose	new data flow	Status	
Desc...tion	This is a new		

Similarly, we also add the File->Input->**tFileInputDelimited** transformation to read the data coming from the aircraft-manufaturerinfo-lookup.csv file.



Talend Open Studio for Data Integration (8.0.1.20211109_1610) | DW-training (Connection: Local)

LOCAL: DW-training
- Job Designs
  - DW_job 0.1
- Contexts
- Code
- SQL Templates
- Metadata
  - Db Connections
    - AIMS 0.1
  - File delimited
  - File positional
  - File regex
  - File XML

Palette: ELT, ESB, File, Input: tApacheLogInput, tFileInputARFF, tFileInputDelimited, tFileInputExcel, tFileInputLDIF, tFileInputMail, tFileInputMSDelimit..., tFileInputMSPositio..., tFileInputMSXML, tFileInputPositional, tFileInputProperties, tFileInputRaw, tFileInputRegex, tFileInputXML, Management

Job(DW_job 0.1) | Contexts(DW_job) | Component | Run (Job DW_job)

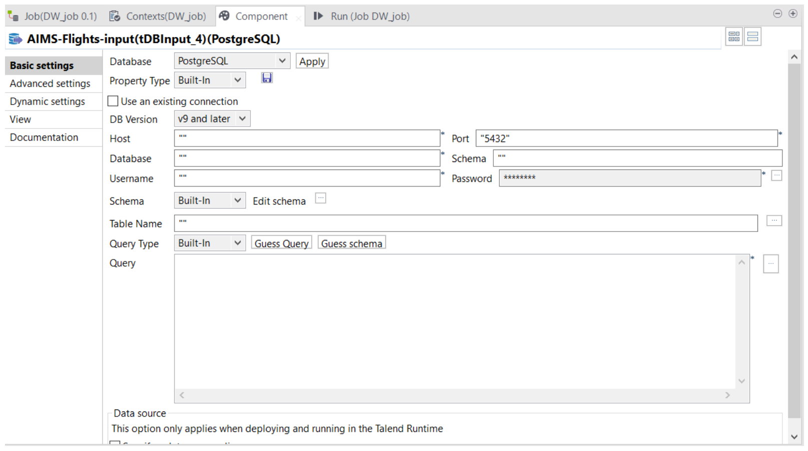
DW_job 0.1

Main | Extra | Stats & Logs | Version

Name	DW_job		
Author	user@talend.com	Version	0.1
Creation	9/13/22, 3:00 PM	Modi...tion	9/13/22, 3:00 PM
Purpose	new data flow	Status	
Desc...tion	This is a new		

a) *Parameterize table input component:*

After clicking the **tPostgresqlInput** component (the inner green square, not the outer grey one), in the control pane below, under the **Component** tab, the configuration to parametrize the added component will open, as seen in the figure below.

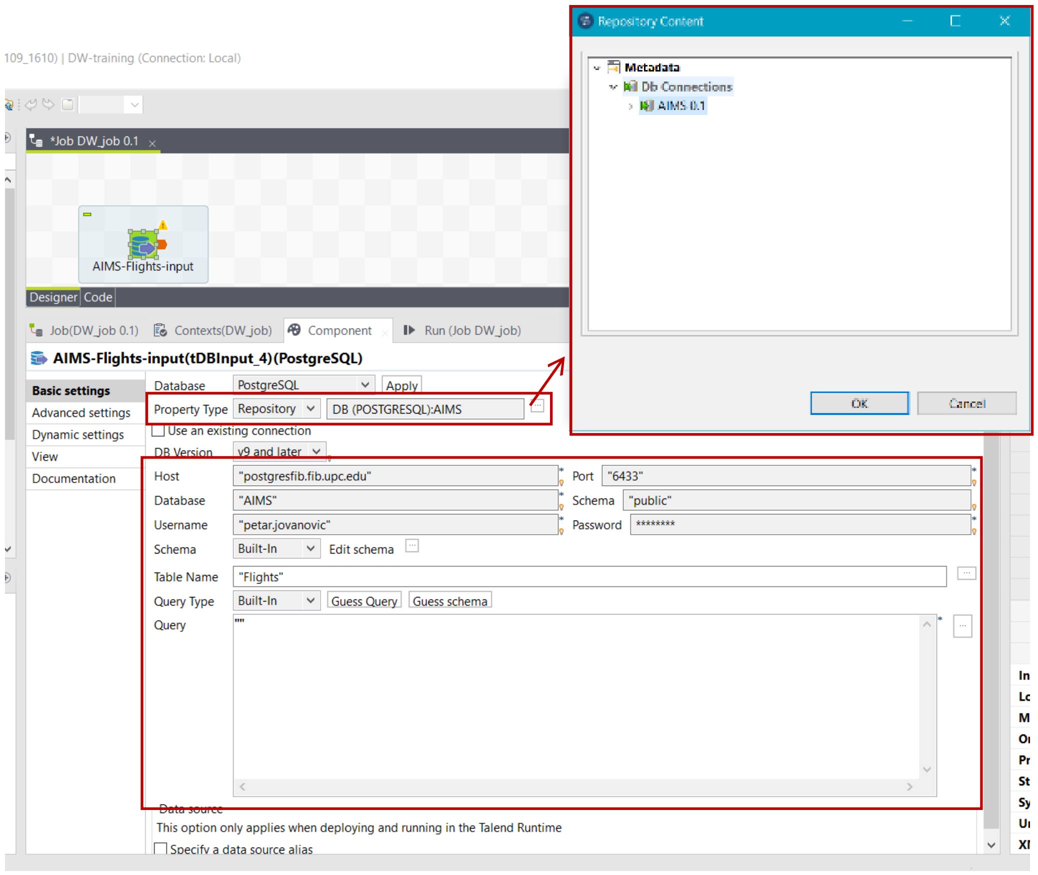


AIMS-Flights-input(tDBInput_4)(PostgreSQL)

Database	PostgreSQL		
Property Type	Built-In		
☐ Use an existing connection			
DB Version	v9 and later		
Host	""	Port	"5432"
Database	""	Schema	""
Username	""	Password	********
Schema	Built-In		
Table Name	""		
Query Type	Built-In		
Query			

Data source
This option only applies when deploying and running in the Talend Runtime

In the **Basic settings** section we can define a new DB connection or use a previously created one. We will use the connection to the AIMS DB that we previously added to the metadata. For this, in the **Property type** section we choose the **Repository** option and then by clicking … button open the **Repository Content** window where we can locate our Db Connection (AIMS), and click **OK**. You will see how the connection parameters below are then filled in automatically from the stored metadata.



AIMS-Flights-input(tDBInput_4)(PostgreSQL)

Database: PostgreSQL
Property Type: Repository — DB (POSTGRESQL):AIMS
DB Version: v9 and later
Host: "postgresfib.fib.upc.edu" Port: "6433"
Database: "AIMS" Schema: "public"
Username: "petar.jovanovic" Password: ********
Schema: Built-In
Table Name: "Flights"
Query Type: Built-In
Query: ""

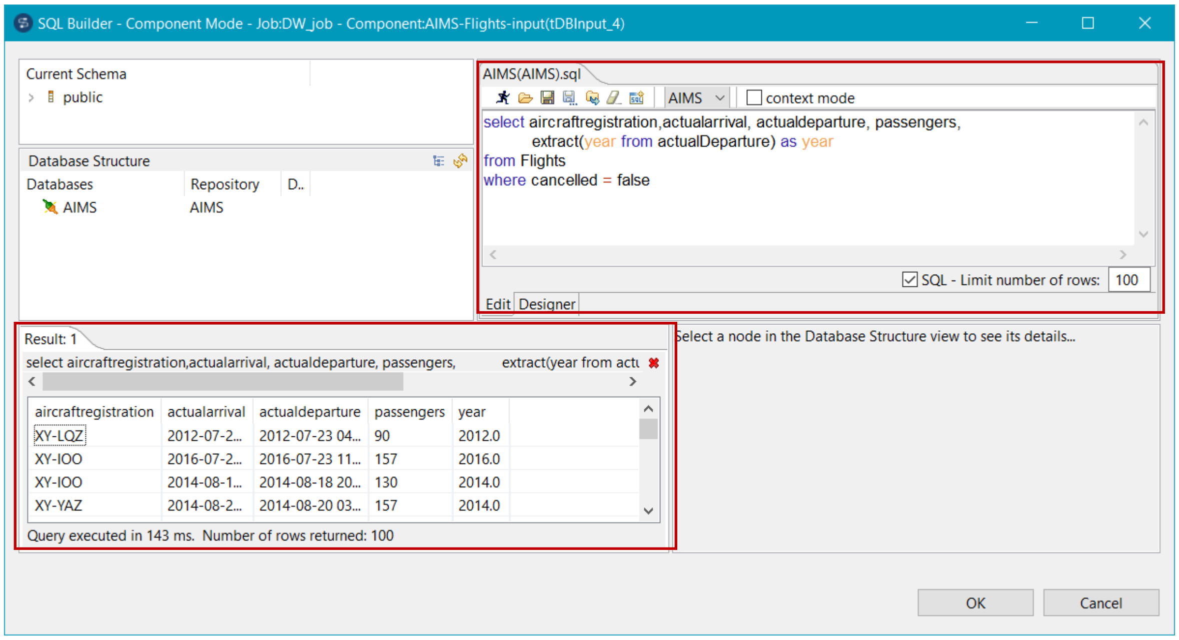
Data source
This option only applies when deploying and running in the Talend Runtime
Specify a data source alias

We can as well define the table name from the given DB (in this case we write “Flights”).

The next step is to obtain the required columns from the given table. For that in the **Query** section we click on … and open the SQL Builder window.

There we can write the SQL query[3] to obtain only the needed data from the table Flights. Notice that since we are calculating the flight hours, we are interested only in the flights that were not cancelled, thus we also add a WHERE clause with the condition “cancelled = false” to the SQL query. We can execute the given query and visualize the sample rows from the table (100 by default), by clicking the running human button. We can use SQL Builder to test several queries and see what kind of data it is giving us. Once we are satisfied with the query to extract the data, we can click OK (See figure below).

[3] As we do not need the data of all the columns to be read we can specify only those needed (avoid using * to get the whole table) and/or apply operations to extract parts of other attributes (e.g., extract (year from actualDeparture)).



```
select aircraftregistration,actualarrival, actualdeparture, passengers,
        extract(year from actualDeparture) as year
from Flights
where cancelled = false
```

☑ SQL - Limit number of rows: 100

Result: 1

select aircraftregistration,actualarrival, actualdeparture, passengers,   extract(year from actu

aircraftregistration	actualarrival	actualdeparture	passengers	year
XY-LQZ	2012-07-2...	2012-07-23 04...	90	2012.0
XY-IOO	2016-07-2...	2016-07-23 11...	157	2016.0
XY-IOO	2014-08-1...	2014-08-18 20...	130	2014.0
XY-YAZ	2014-08-2...	2014-08-20 03...	157	2014.0

Query executed in 143 ms. Number of rows returned: 100

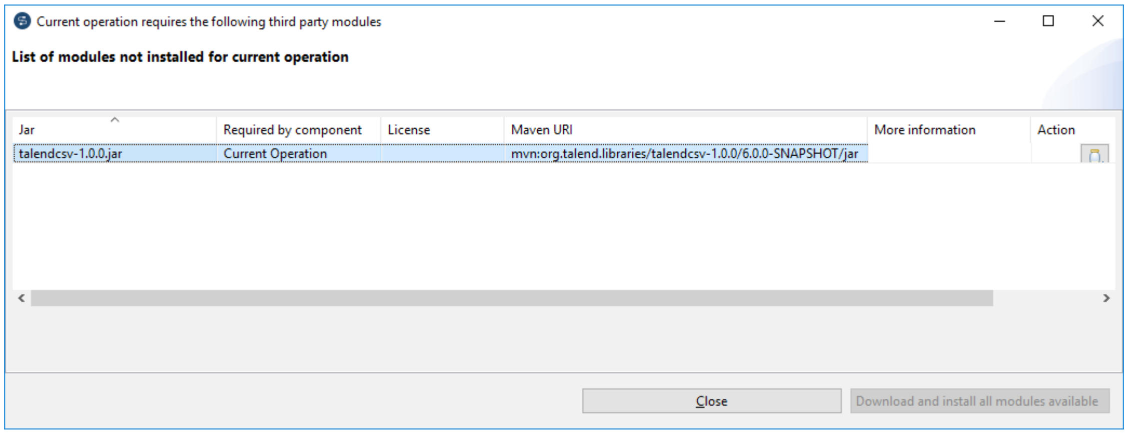
Select a node in the Database Structure view to see its details...

We can see then that the provided query is placed inside the **Query** section of the component configuration. After this, in order to define the output schema, automatically from the given query, you should click on the **Guess schema** option that will automatically list the output columns from the query with data types *guessed* from executing a sample of the input data.

Importantly, notice that this step would require an external 3rd party jar library (**talendcsv**) to be installed.

Installing 3rd party libraries in Talend

In some cases, if an external library is missing in Talend, and it is not included in the installation package, it must be downloaded and installed manually. Here we will show this process for the **talendcsv** library, used for handling csv files as well as obtaining schema information automatically. If you click on the **Guess schema** option like in the previous step, you will initially get the following window :

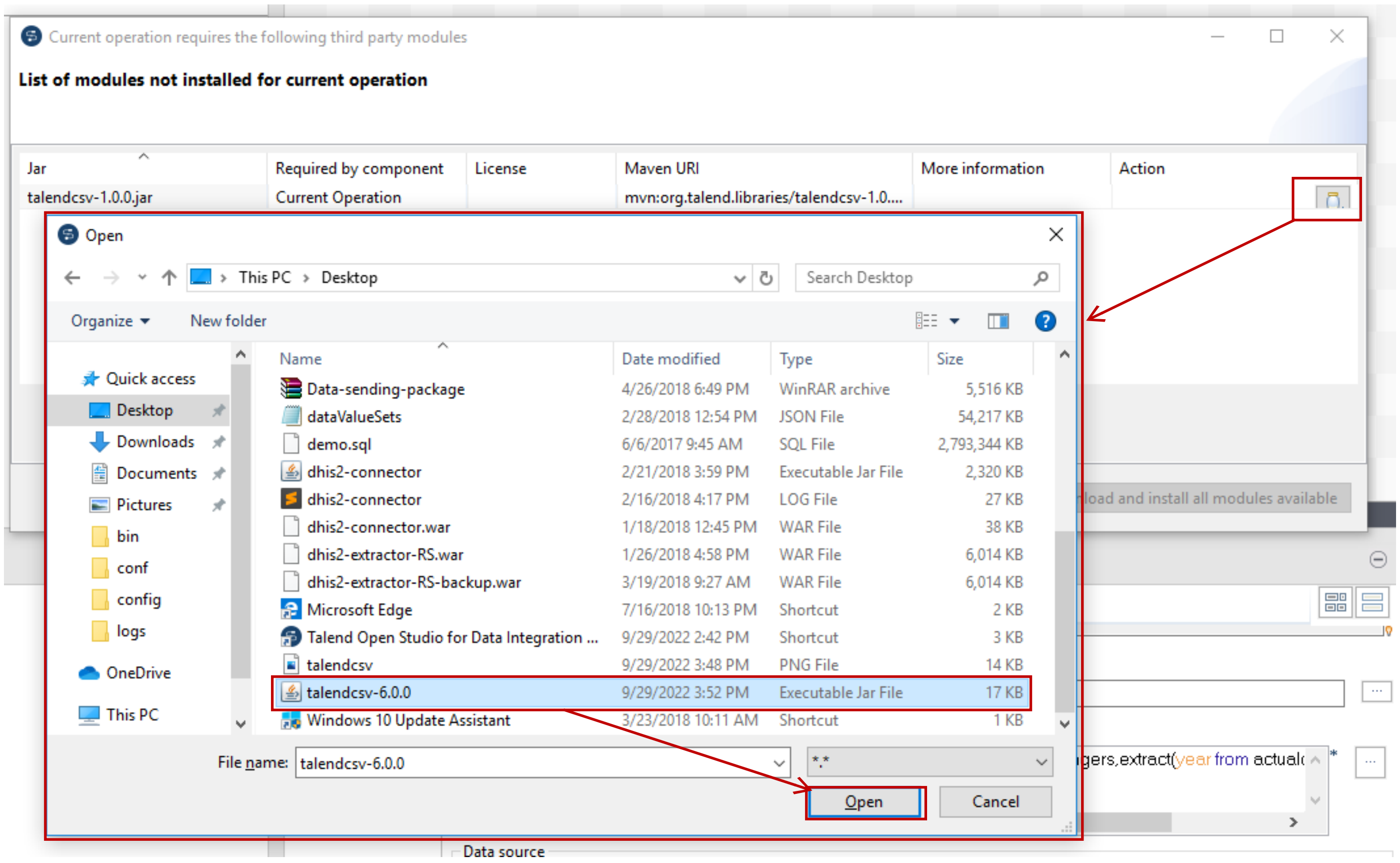

Current operation requires the following third party modules
Close
Download and install all modules available

List of modules not installed for current operation

Jar	Required by component	License	Maven URI	More information	Action
talendcsv-1.0.0.jar	Current Operation		mvn:org.talend.libraries/talendcsv-1.0.0/6.0.0-SNAPSHOT/jar		

Notice that in this case, there is no option to download and install this module automatically, like in the case of PostgreSQL JDBC library. In such case, we must search for the external library in the online maven repository[4], download it and then point to it from Talend, as shown in the figure below.

[4] https://artifacts-oss.talend.com/nexus/content/repositories/TalendOpenSourceRelease/org/talend/libraries/talendcsv/6.0.0/talendcsv-6.0.0.jar


Current operation requires the following third party modules

List of modules not installed for current operation

Jar	Required by component	License	Maven URI	More information	Action
talendcsv-1.0.0.jar	Current Operation		mvn:org.talend.libraries/talendcsv-1.0....		

Open

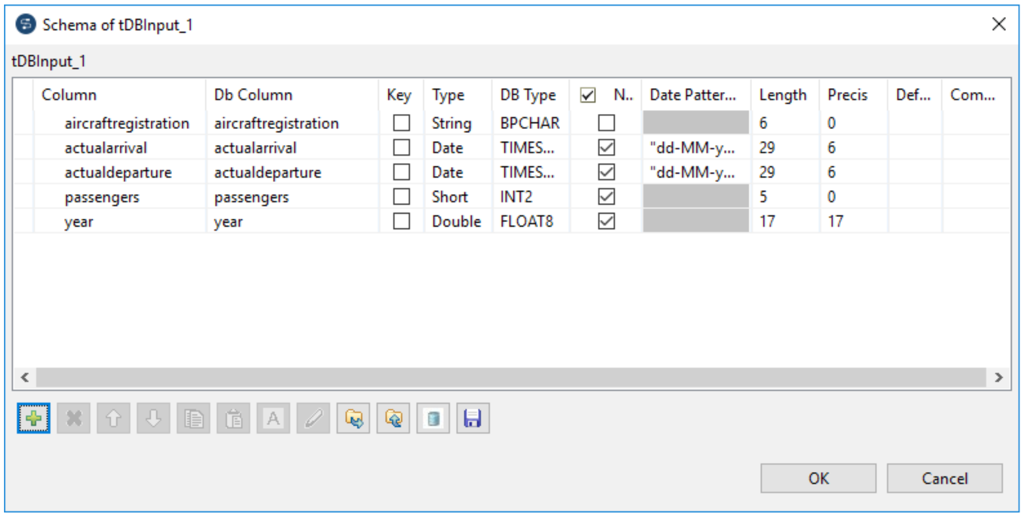
This PC › Desktop

Name	Date modified	Type	Size
Data-sending-package	4/26/2018 6:49 PM	WinRAR archive	5,516 KB
dataValueSets	2/28/2018 12:54 PM	JSON File	54,217 KB
demo.sql	6/6/2017 9:45 AM	SQL File	2,793,344 KB
dhis2-connector	2/21/2018 3:59 PM	Executable Jar File	2,320 KB
dhis2-connector	2/16/2018 4:17 PM	LOG File	27 KB
dhis2-connector.war	1/18/2018 12:45 PM	WAR File	38 KB
dhis2-extractor-RS.war	1/26/2018 4:58 PM	WAR File	6,014 KB
dhis2-extractor-RS-backup.war	3/19/2018 9:27 AM	WAR File	6,014 KB
Microsoft Edge	7/16/2018 10:13 PM	Shortcut	2 KB
Talend Open Studio for Data Integration ...	9/29/2022 2:42 PM	Shortcut	3 KB
talendcsv	9/29/2022 3:48 PM	PNG File	14 KB
talendcsv-6.0.0	9/29/2022 3:52 PM	Executable Jar File	17 KB
Windows 10 Update Assistant	3/23/2018 10:11 AM	Shortcut	1 KB

File name: talendcsv-6.0.0

Once the 3rd party library is successfully installed in Talend, you should be able to use the given functionality.

In this case, if we click again the **Guess Schema** option, you should be able to see the automatically *guessed* schema with columns and their datatypes, as seen in the figure below.

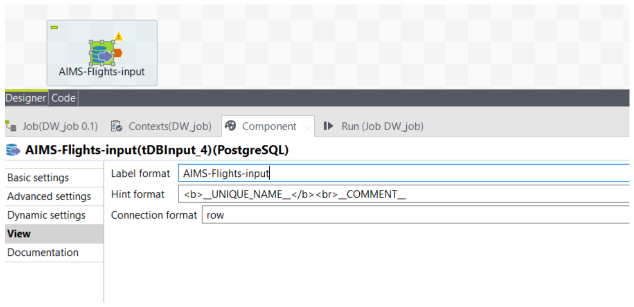

Schema of tDBInput_1

tDBInput_1

Column	Db Column	Key	Type	DB Type	N..	Date Patter...	Length	Precis	Def...	Com...
aircraftregistration	aircraftregistration	☐	String	BPCHAR	☐		6	0		
actualarrival	actualarrival	☐	Date	TIMES...	☑	"dd-MM-y...	29	6		
actualdeparture	actualdeparture	☐	Date	TIMES...	☑	"dd-MM-y...	29	6		
passengers	passengers	☐	Short	INT2	☑		5	0		
year	year	☐	Double	FLOAT8	☑		17	17		

You can edit it, if necessary. Click OK and it will store the schema metadata for this component.

In addition, as part of the **Component** configuration tab, under the **View** section (**Label format**) we can also change the name of the component and give it a more project-specific name. See the figure below.



AIMS-Flights-input(tDBInput_4)(PostgreSQL)

Label format: AIMS-Flights-input
Hint format: <b>__UNIQUE_NAME__</b>
__COMMENT__
Connection format: row

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

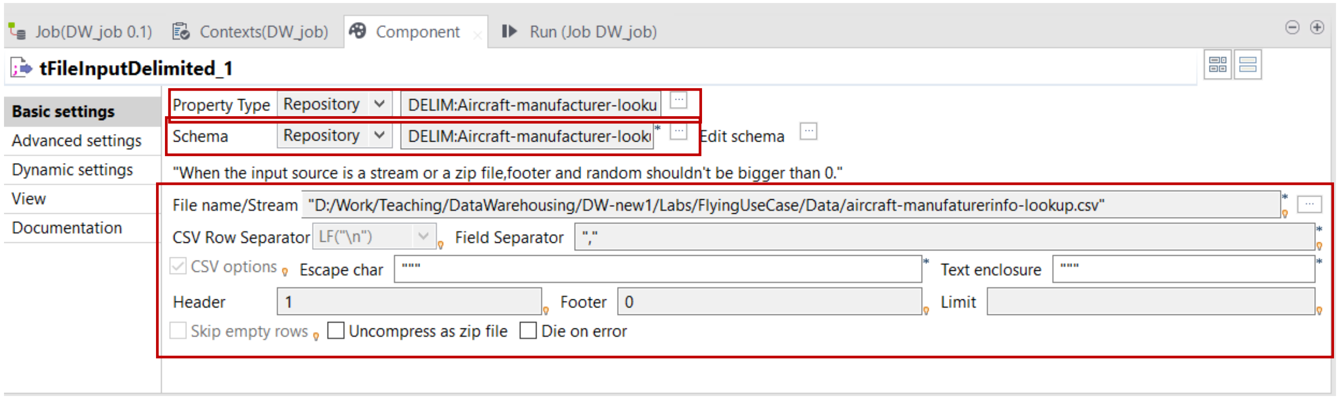
b) Parameterize tFileInputDelimited[5] component:

Defining CSV file/schema metadata

Just like with the DB connection, Talend allows us to centralize the definition of the metadata needed to connect and read CSV files (i.e., delimited file). Before you start parametrizing the tFileInputDelimited component, please follow carefully the steps explained in the Talend web page[6] to define the connection to the Aircraft manufacturer info lookup file.

(1) As with any other component, we can define the name of the step by entering the desired name in the **Label format** text box inside the **View** section of the **Component** tab.

(2) In the **Basic settings** section of the **Component** tab we can then define the configuration to read the data from the CSV file. Since we previously created a centralized metadata to treat this CSV file, we should simply select this metadata under **Property Type** and **Schema** sections from the **Repository**. Once we do this, the configuration and schema will be filled in automatically. Alternatively, we can use **Built-in Property type** and **Schema**, and create a one-time configuration for this component. The advantage of using the centralized metadata is that we can easily reuse them various times inside the same project. See the figure below.



tFileInputDelimited_1

Property Type: Repository — DELIM:Aircraft-manufacturer-looku
Schema: Repository — DELIM:Aircraft-manufacturer-looki — Edit schema

"When the input source is a stream or a zip file,footer and random shouldn't be bigger than 0."

File name/Stream: "D:/Work/Teaching/DataWarehousing/DW-new1/Labs/FlyingUseCase/Data/aircraft-manufaturerinfo-lookup.csv"
CSV Row Separator: LF("\n") — Field Separator: ","
☑ CSV options — Escape char: """ — Text enclosure: """
Header: 1 — Footer: 0 — Limit:
☐ Skip empty rows ☐ Uncompress as zip file ☐ Die on error

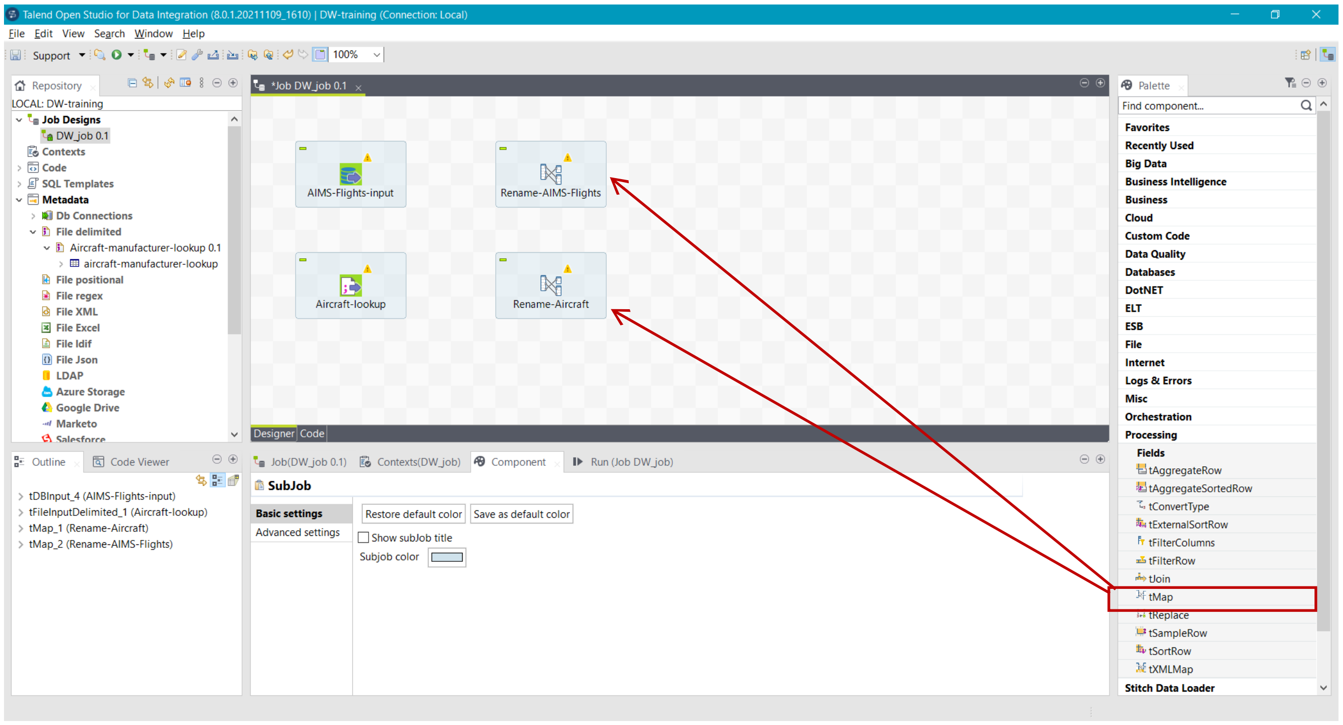
4. **Adding the data transformation components of the ETL process:**

After we defined the input data components to extract data we need to define the transformation steps of the ETL process.

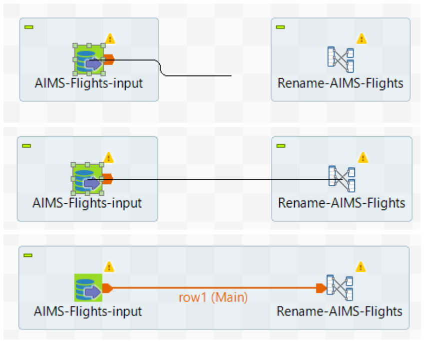
(1) First, for each input data step we can add a *rename* step that will assure that all the field names in

[5] https://help.talend.com/r/en-US/8.0/delimited/tfileinputdelimited
[6] https://help.talend.com/r/en-US/8.0/studio-user-guide-esb/centralizing-file-delimited-metadata

the process are unique. This step can be important as various independent sources may have the same names for the fields with different semantics. From the **Palette** we choose the Processing->**tMap**[7] component and drag it to the designer canvas. See figure below.

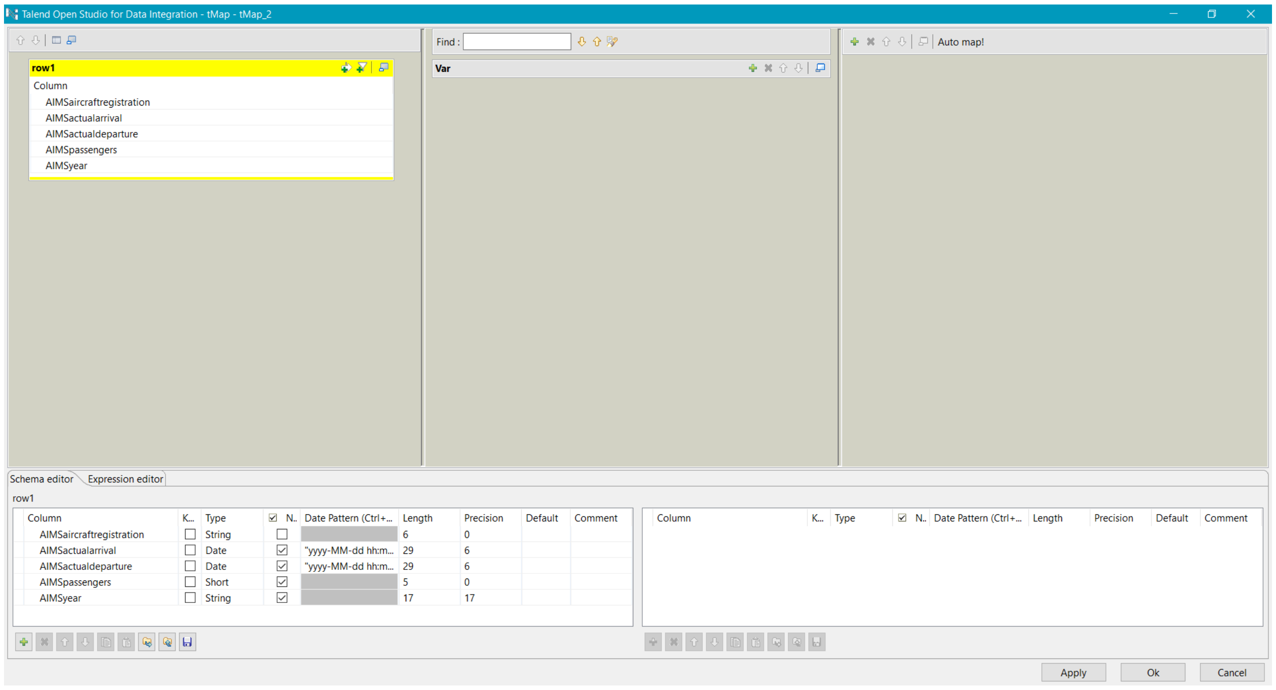


(2) To insert the edge from the table input or csv file input components to the rename component (or between any other two components in the canvas), we click on the first component and then a small red arrow will appear on its right ( ). When we click and hold the red arrow it will activate edge creation and we just need to drag it towards the component to which we want to connect. Once we release the mouse button it will establish the data flow edge between the two components. See the figures below. Another interesting consequence of this action is that these two components will become part of the same *subjob (sub data flow)*.



(3) To parameterize the **tMap** component for renaming the columns, we need to double click the **tMap** component which opens the **tMap** configuration window as seen in the figure below. Then, inside the **Schema editor** tab below we simply need to click on the column we want to rename and give it another name. For example, to distinguish the columns coming from different sources, we can add as a prefix the name of the source (e.g., AIMS). Besides renaming inside the same Schema editor we can also alter the metadata of input fields (i.e., data type, length, precision, or format of the date). After we finish, we click **Apply** and **OK** to save the configuration.

[7] tMap component has very rich semantics: For more details refer to: https://help.talend.com/r/en-US/8.0/tmap/tmap-component



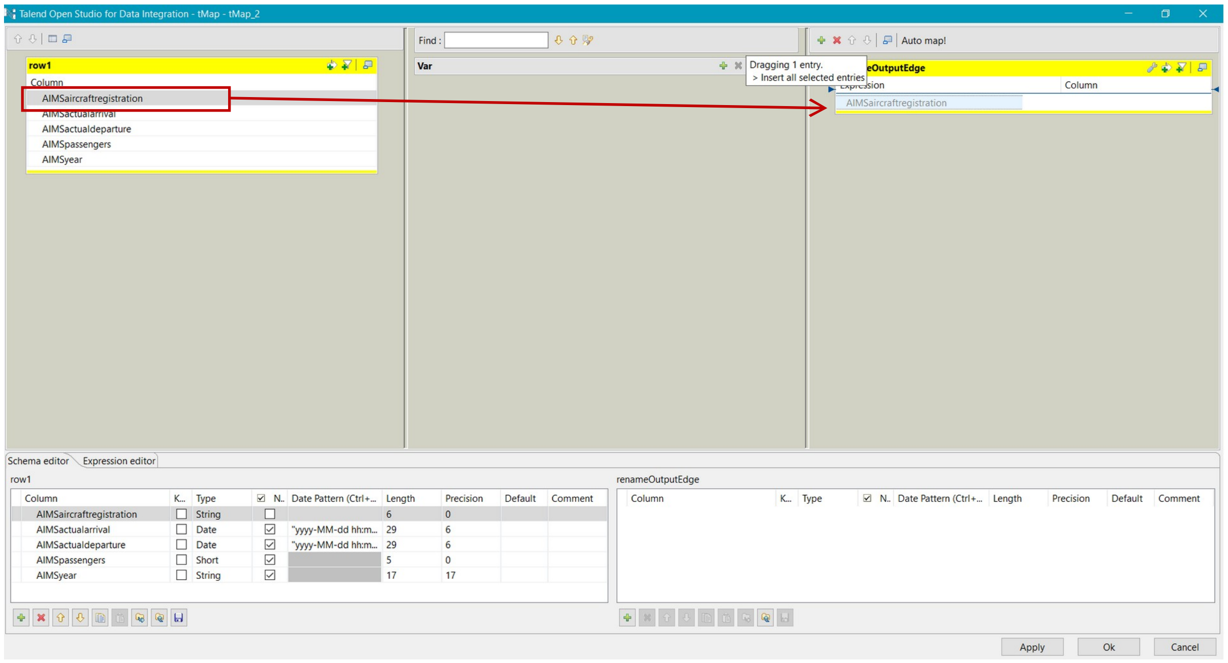
row1

Column
AIMSaircraftregistration
AIMSactualarrival
AIMSactualdeparture
AIMSpassengers
AIMSyear

Column	K...	Type	N..	Date Pattern (Ctrl+...	Length	Precision	Default	Comment
AIMSaircraftregistration	☐	String	☐		6	0		
AIMSactualarrival	☐	Date	✓	"yyyy-MM-dd hh:m...	29	6		
AIMSactualdeparture	☐	Date	✓	"yyyy-MM-dd hh:m...	29	6		
AIMSpassengers	☐	Short	✓		5	0		
AIMSyear	☐	String	✓		17	17		

Column	K...	Type	N..	Date Pattern (Ctrl+...	Length	Precision	Default	Comment

Later on, after we connect the tMap component to the next component in the data flow (i.e., after completing step 4) below), we need to come back and map the input schema to the output schema, by dragging only those attributes that we want to pass to the next component.



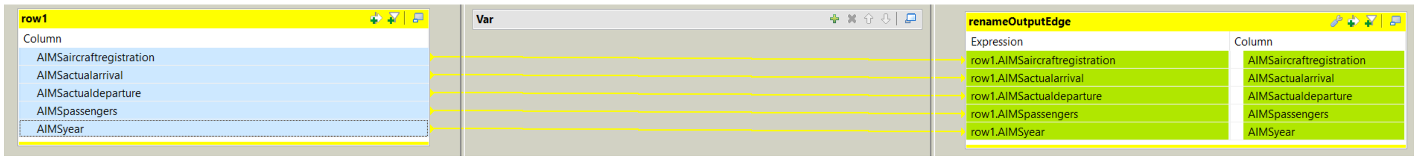
row1

Column	K...	Type	N..	Date Pattern (Ctrl+...	Length	Precision	Default	Comment
AIMSaircraftregistration	☐	String	☐		6	0		
AIMSactualarrival	☐	Date	✓	"yyyy-MM-dd hh:m...	29	6		
AIMSactualdeparture	☐	Date	✓	"yyyy-MM-dd hh:m...	29	6		
AIMSpassengers	☐	Short	✓		5	0		
AIMSyear	☐	String	✓		17	17		

renameOutputEdge

Column	K...	Type	N..	Date Pattern (Ctrl+...	Length	Precision	Default	Comment

In this case we map all 5 columns to the output



Column
AIMSaircraftregistration
AIMSactualarrival
AIMSactualdeparture
AIMSpassengers
AIMSyear

Expression	Column
row1.AIMSaircraftregistration	AIMSaircraftregistration
row1.AIMSactualarrival	AIMSactualarrival
row1.AIMSactualdeparture	AIMSactualdeparture
row1.AIMSpassengers	AIMSpassengers
row1.AIMSyear	AIMSyear

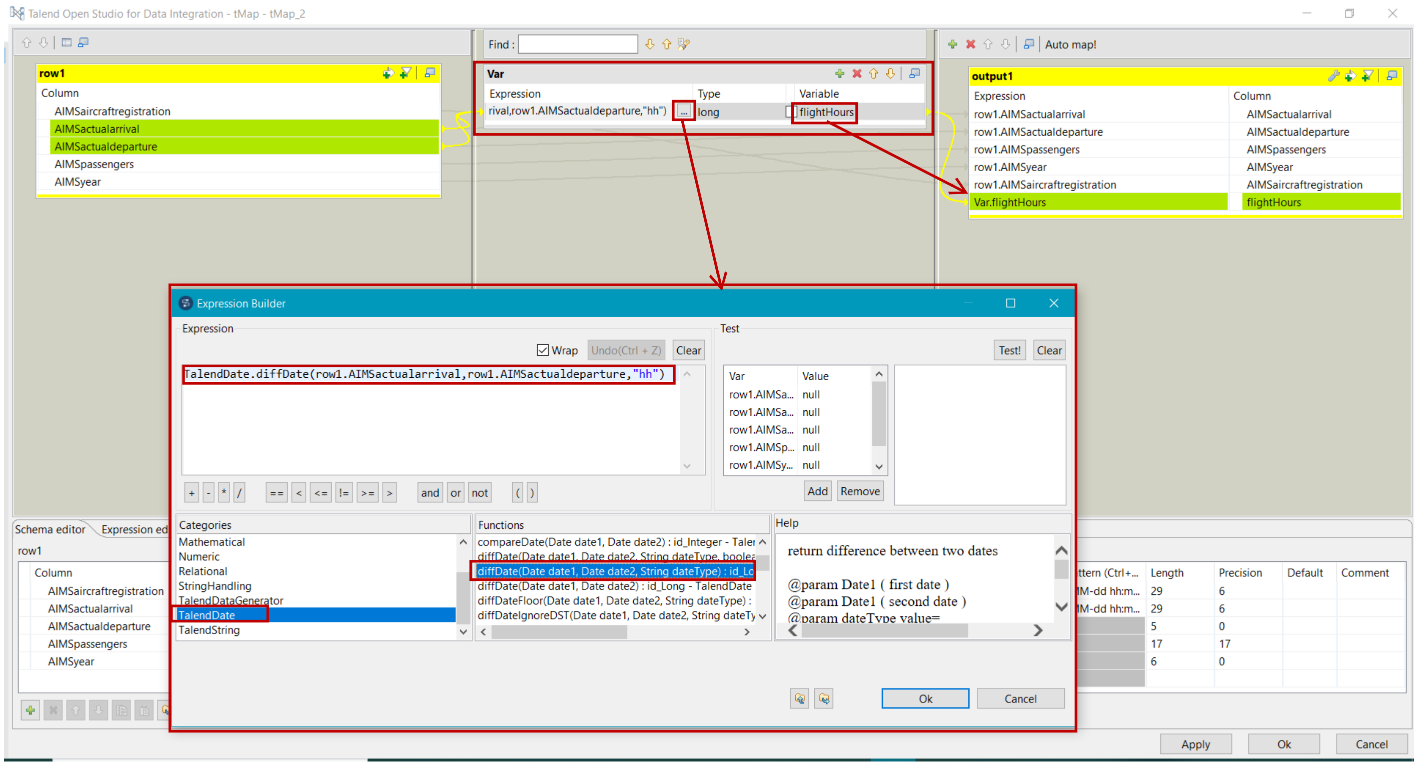
Notice that with this functionality we can do projections of the columns, omitting the columns that are no longer needed in the flow.

Notice that this way we performed renaming at the input of the tMap component (i.e., in the left hand side

Schema editor), which means that the renamed columns will be propagated to any output in the rest of the flow. Similarly, instead of doing it at the input, we could as well do the renaming at the output of tMap (i.e., in the right hand side **Schema editor**), but only after we connect the tMap component to another component(s) in the ETL flow and have the columns properly mapped to that output. This way, the renaming should be done for each output of the tMap (in case there is more than one), and the columns can get different names different outputs.

The tMap component has very rich semantics and we will use it more throughout this tutorial. For more details, refer to: https://help.talend.com/r/en-US/8.0/tmap/tmap-component

Indeed, we will take advantage and derive a new variable with an expression to calculate the flightHours attribute as required in KPI 1. Inside the same configuration we will add the expression (made out of input attributes and define the new variable, which we will then pass to the output. See the figure below. Inside the Expression Builder window, in this case, we will use a special Talend operator for dealing with the dates (**TalendDate**). More precisely to calculate the difference (**diffDate**) between the timestamp of the arrival and the timestamp of the departure in terms of hours (**"hh"**), in order to calculate the flight hours of each flight. Notice that this way, we use the same **tMap** component to do the renaming of the columns and deriving the new attribute - **flightHours**.


Talend Open Studio for Data Integration - tMap - tMap_2

row1

Column
AIMSaircraftregistration
AIMSactualarrival
AIMSactualdeparture
AIMSpassengers
AIMSyear

Var

Expression	Type	Variable
rival,row1.AIMSactualdeparture,"hh")	long	flightHours

output1

Expression	Column
row1.AIMSactualarrival	AIMSactualarrival
row1.AIMSactualdeparture	AIMSactualdeparture
row1.AIMSpassengers	AIMSpassengers
row1.AIMSyear	AIMSyear
row1.AIMSaircraftregistration	AIMSaircraftregistration
Var.flightHours	flightHours

Expression Builder

```
TalendDate.diffDate(row1.AIMSactualarrival,row1.AIMSactualdeparture,"hh")
```

Var	Value
row1.AIMSa...	null
row1.AIMSa...	null
row1.AIMSa...	null
row1.AIMSp...	null
row1.AIMSy...	null

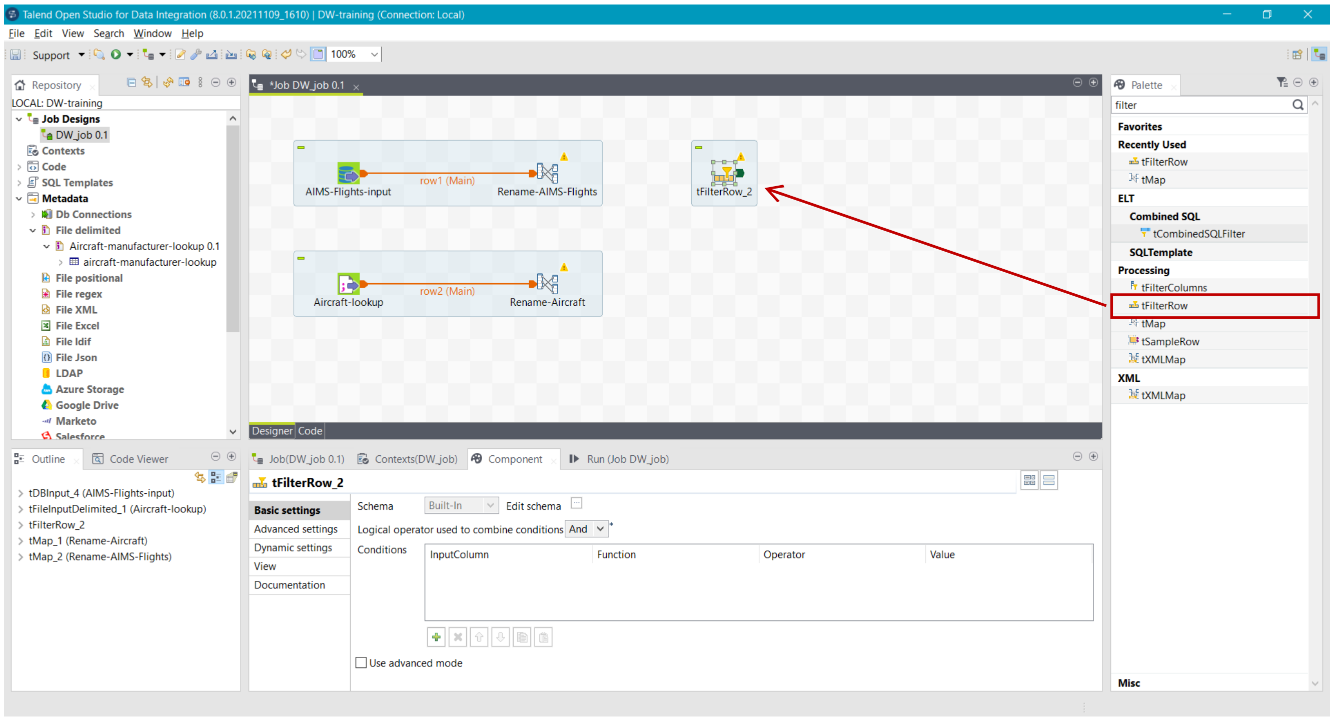
Categories: Mathematical, Numeric, Relational, StringHandling, TalendDataGenerator, TalendDate, TalendString

Functions:
compareDate(Date date1, Date date2) : id_Integer - Talen
diffDate(Date date1, Date date2, String dateType, boolea
diffDate(Date date1, Date date2, String dateType) : id_Lo
diffDate(Date date1, Date date2) : id_Long - TalendDate
diffDateFloor(Date date1, Date date2, String dateType) :
diffDateIgnoreDST(Date date1, Date date2, String dateTy

Help:
return difference between two dates

@param Date1 (first date)
@param Date1 (second date)
@param dateType value=

(4) Then, we need to add a *filtering* step for the restriction defined in the KPI 1(i.e., *passengers > 50*). From the **Processing** category of the **Palette** choose Processing->**tFilterRow** component and drag and drop it to the designer canvas as shown in the figure. Before parameterizing it, we need to connect it to the previous **Rename-AIMS-Flights** step and configure the **tMap** as explained previously.



tFilterRow_2 — Basic settings

Schema: Built-In — Edit schema

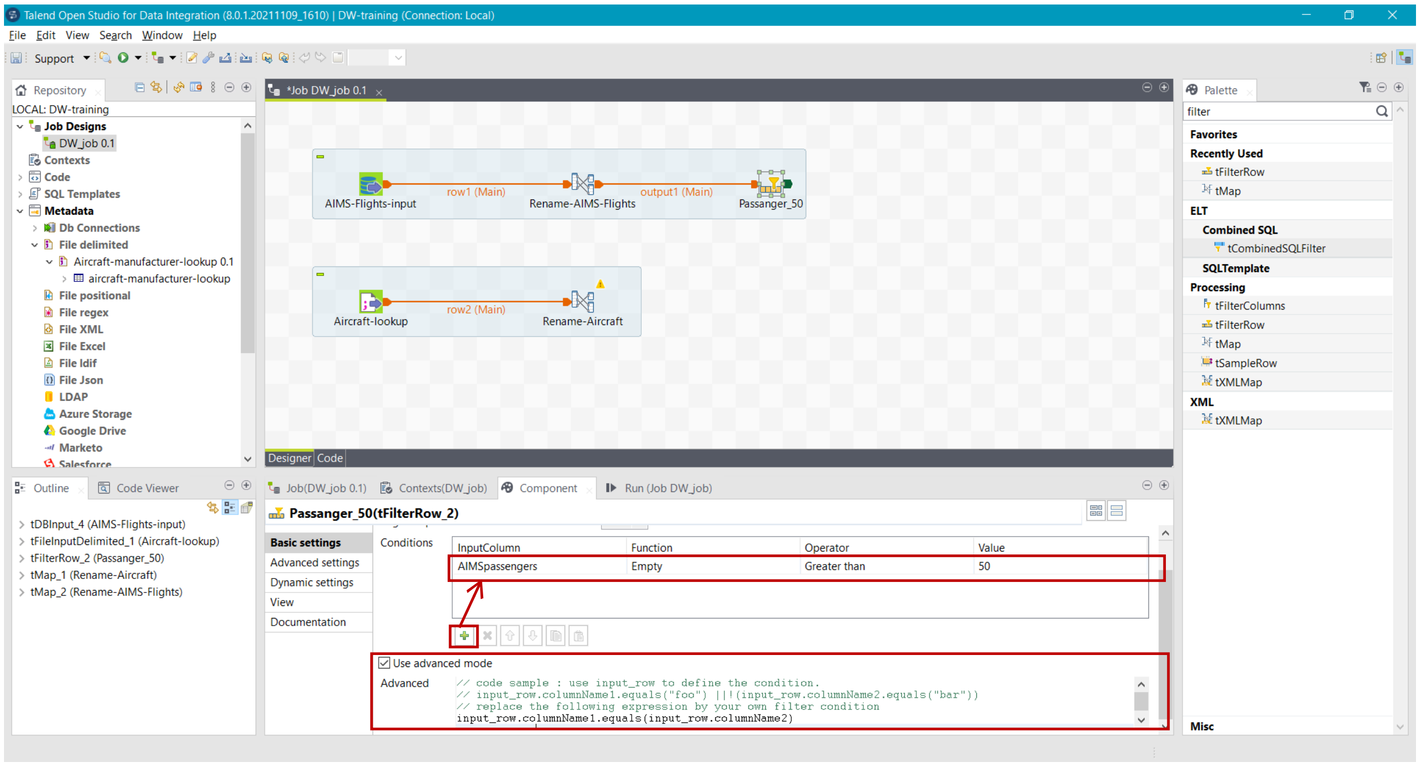
Logical operator used to combine conditions: And

Conditions:

InputColumn	Function	Operator	Value

☐ Use advanced mode

(5) We then need to parameterize the **tFilterRow** component considering the restriction provided in the input KPI 1. See figure below.



Passanger_50(tFilterRow_2)

Conditions:

InputColumn	Function	Operator	Value
AIMSpassengers	Empty	Greater than	50

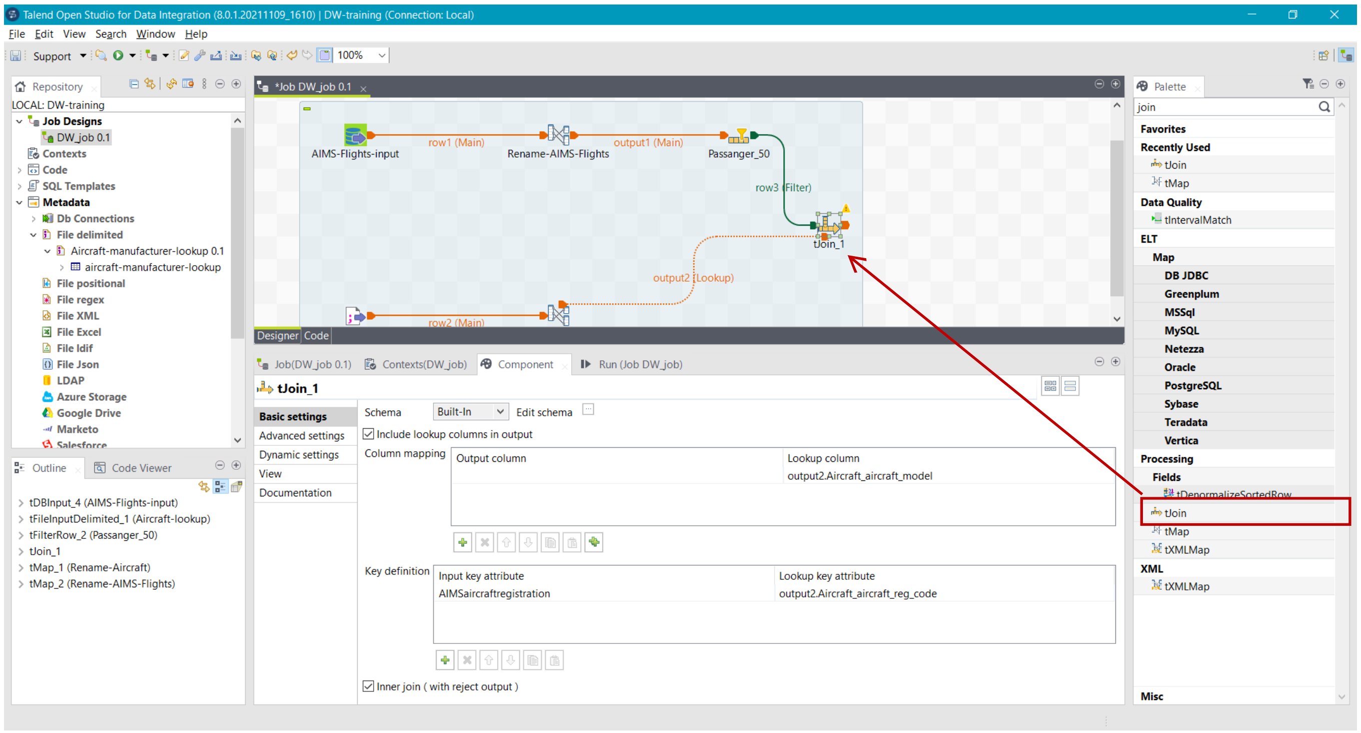
☑ Use advanced mode

Advanced:

```
// code sample : use input_row to define the condition.
// input_row.columnName1.equals("foo") ||!(input_row.columnName2.equals("bar"))
// replace the following expression by your own filter condition
input_row.columnName1.equals(input_row.columnName2)
```

Important note: *while tFilterRow provides you with typical predefined predicates to filter the values of input fields, to implement more complex Filtering operations, you can use the advanced mode and write your filters in Java. In this case, we will leave it unselected.*

(6) Then we add the *Join operations* to relate data coming from the two defined data sources (i.e., AIMS Flights table and Manufacturer lookup table from the CSV file). From the **Processing** category of the **Palette** we choose the **tJoin** component and drag it to the designer canvas as shown in the figure below. As before, we need to connect it to the previous steps in the flow and in addition configure **tMap** of the **Rename-Aircraft** step such that the correct columns are mapped to the **output2**.



tJoin_1

Basic settings

Schema: Built-In — Edit schema

☑ Include lookup columns in output

Column mapping

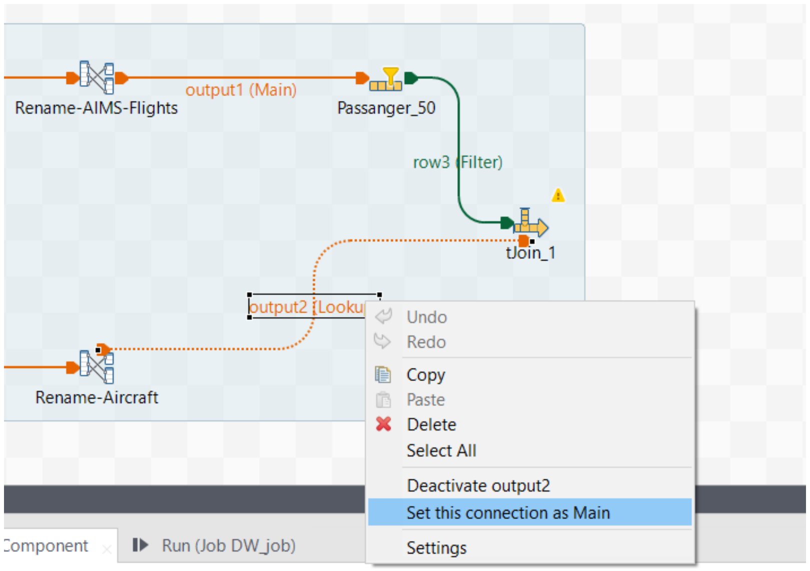
Output column	Lookup column
	output2.Aircraft_aircraft_model

Key definition

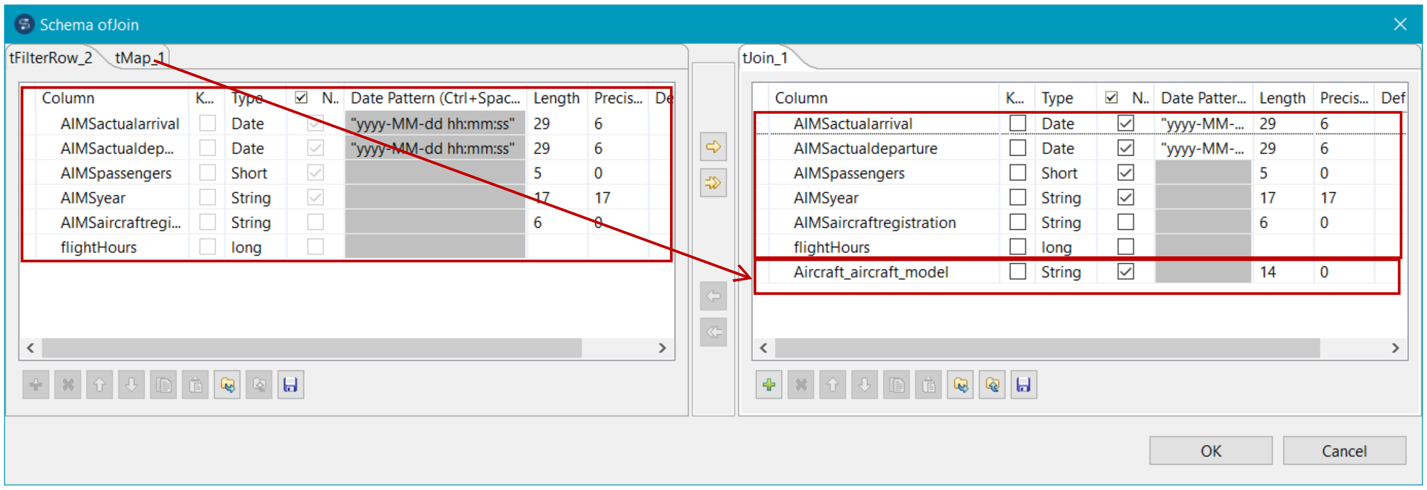
Input key attribute	Lookup key attribute
AIMSaircraftregistration	output2.Aircraft_aircraft_reg_code

☑ Inner join (with reject output)

(7) Then we parameterize the tJoin step to properly relate in this case rows coming AIMS Flight table (**Main** connection) and rows coming from Aircraft Manufacturer Lookup CSV file (**Lookup** connection). By default, the first connected step is **Main** and the second **Lookup**, but we can change this a posteriori by right-clicking on the connection and choosing the other option (see the figure below). Notice that this serves to optimize the execution of the join, and that the larger data input should be **Main** which is read as a stream, while the Lookup is a smaller and cached in memory.



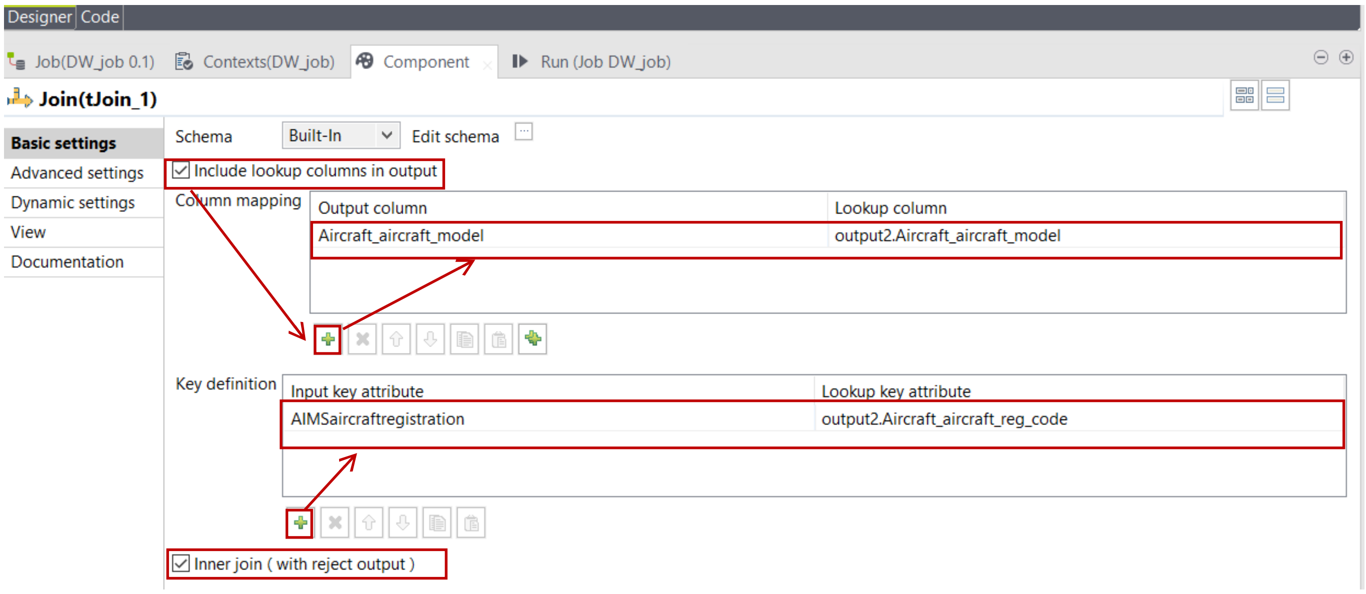
We then need to configure what will be the output schema from this join step. Clicking in the Component tab on **Edit schema …**, opens a window where we can define which attributes from the Main and from the Lookup input step we want to pass on. On the left we see two tabs, one for each input step schema, from which we can select the attributes and pass them to the right under the join output schema. See figure below. We pass all the attributes from the Main input step and only the **Aircraft_aircraft_model** from the Lookup input step.



Column	K...	Type	☑ N..	Date Pattern (Ctrl+Spac...	Length	Precis...	De
AIMSactualarrival	☐	Date	✓	"yyyy-MM-dd hh:mm:ss"	29	6	
AIMSactualdep...	☐	Date	✓	"yyyy-MM-dd hh:mm:ss"	29	6	
AIMSpassengers	☐	Short	✓		5	0	
AIMSyear	☐	String	✓		17	17	
AIMSaircraftregi...	☐	String	☐		6	0	
flightHours	☐	long	☐				

Column	K...	Type	☑ N..	Date Patter...	Length	Precis...	Def
AIMSactualarrival	☐	Date	✓	"yyyy-MM-...	29	6	
AIMSactualdeparture	☐	Date	✓	"yyyy-MM-...	29	6	
AIMSpassengers	☐	Short	✓		5	0	
AIMSyear	☐	String	✓		17	17	
AIMSaircraftregistration	☐	String	☐		6	0	
flightHours	☐	long	☐				
Aircraft_aircraft_model	☐	String	✓		14	0	

Lastly, define the join attributes (**Key definition** section), as well as include the lookup attributes that we want to pass to the output (**Include lookup columns in the output** checkbox, and **Column mapping** section). See the figure below. In this case, as defined in the schema, we only need aircraft model in the output, but we need to additionally add the mapping to the Lookup column. In addition, we also choose to use the Inner join (**Inner join (with reject output)** checkbox), such that the Flights for which we do not have an entry in the lookup file are not passed to the main output. If this option is not selected, the join will act as an *outer join* and by setting Main and Lookup connection we can make it LEFT or RIGHT outer join.



Join(tJoin_1)

Schema: Built-In — Edit schema

☑ Include lookup columns in output

Column mapping

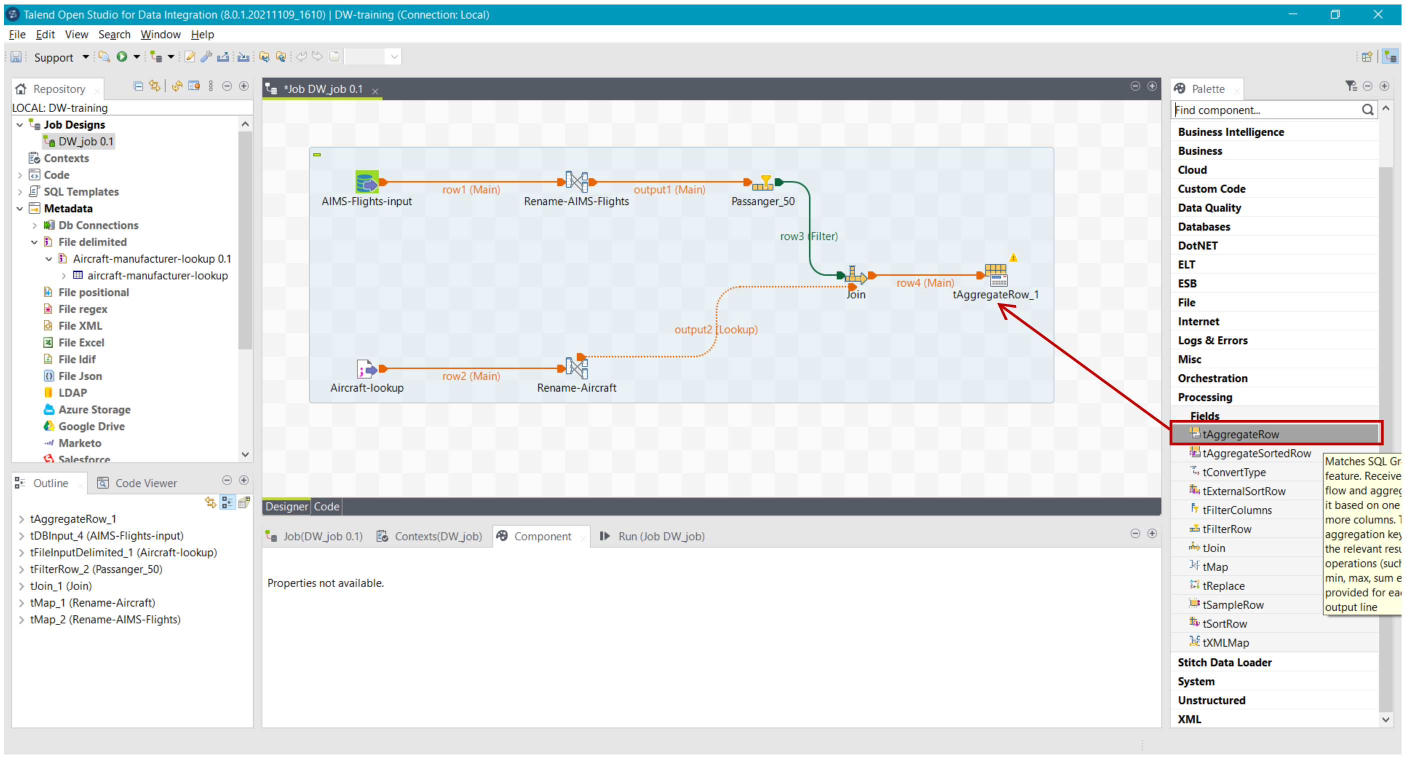
Output column	Lookup column
Aircraft_aircraft_model	output2.Aircraft_aircraft_model

Key definition

Input key attribute	Lookup key attribute
AIMSaircraftregistration	output2.Aircraft_aircraft_reg_code

☑ Inner join (with reject output)

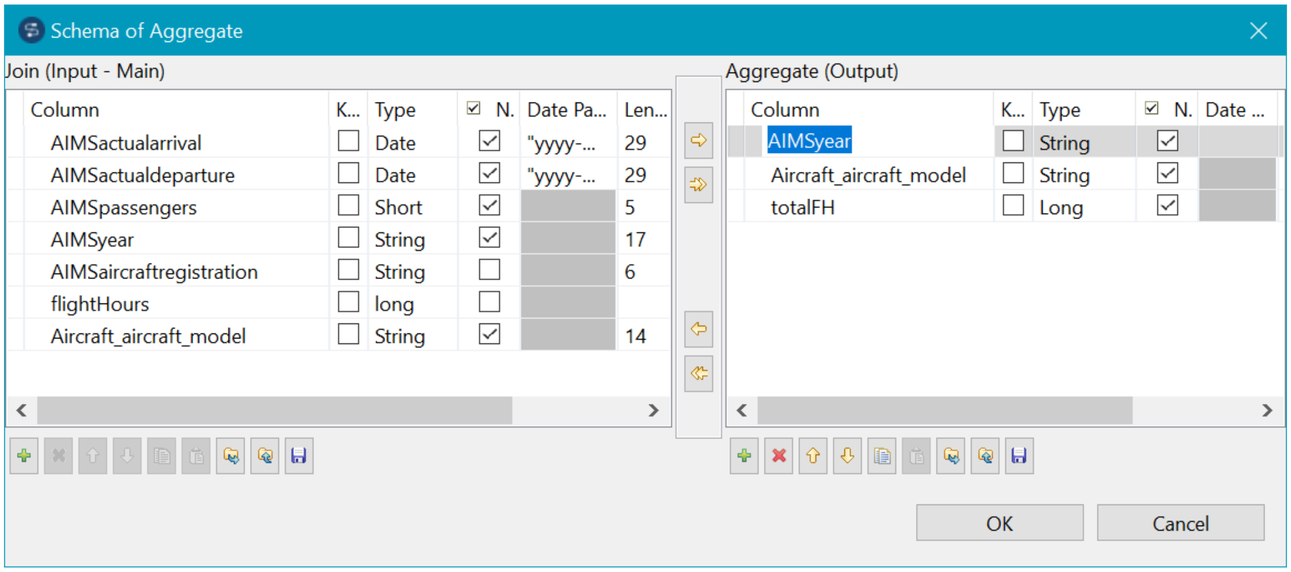
(8) Afterwards, we need to aggregate data considering the given KPI 1. From the **Processing** category of the **Palette** we choose the **tAggregateRow**[8] component and drag it to the designer canvas as shown in the figure below. Afterwards we also need to connect it to the previous step in the data flow - **Join**.



(9) We then parameterize the tAggregateRow component considering KPI 1. First of all, like in the case of Join, we need to define the output schema of the operations, by clicking **Edit schema …** In the schema definition window, we need to select the attributes/columns from the input that we want to send to the output. These would later serve as group by attributes (**Aircraft_aircraft_model** and **AIMSyear**). We then need also to add a new output

[8] Notice that there is also tAggregateSortedRow, which requires that the input rows are sorted on the Group By keys, and can optimize the execution if the input rows already follow such order.

column that will serve to receive the value resulting from the aggregation operation (**totalFH** and choose the **long** as a data type). See figure below.



Schema of Aggregate

Join (Input - Main)

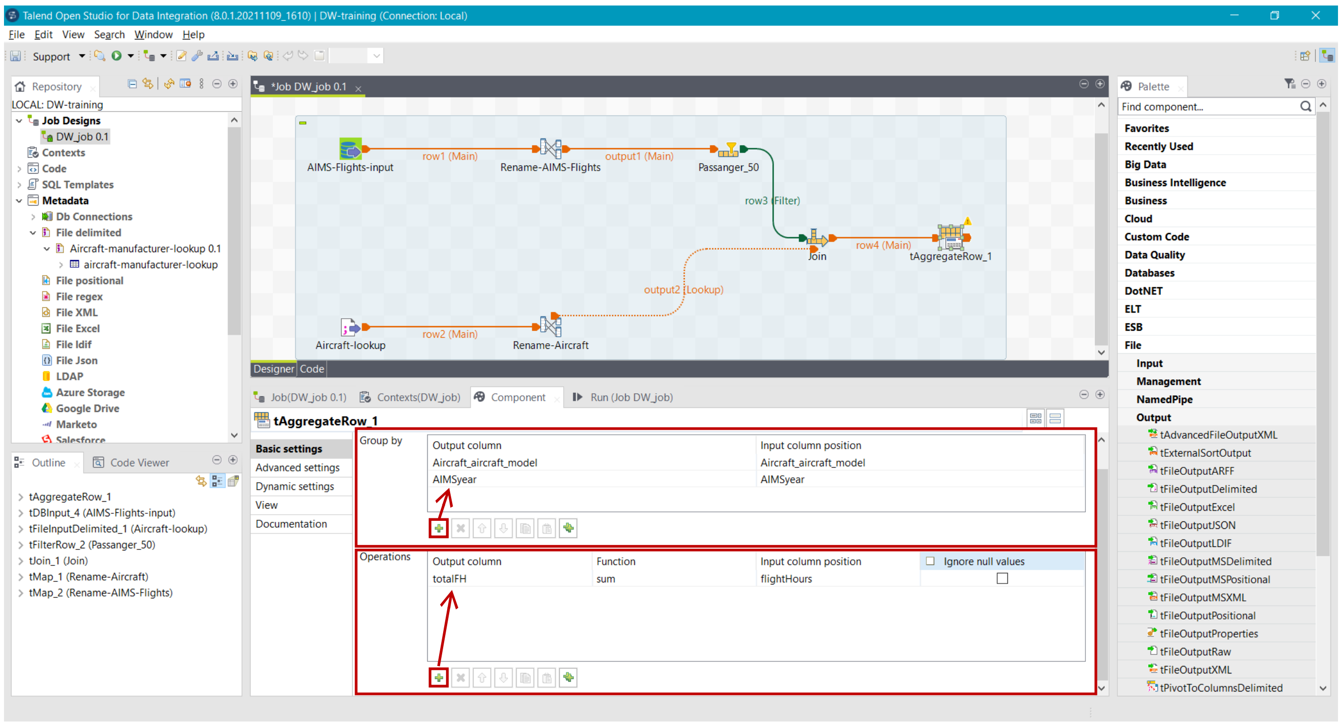
Column	K...	Type	N.	Date Pa...	Len...
AIMSactualarrival	☐	Date	✓	"yyyy-...	29
AIMSactualdeparture	☐	Date	✓	"yyyy-...	29
AIMSpassengers	☐	Short	✓		5
AIMSyear	☐	String	✓		17
AIMSaircraftregistration	☐	String	☐		6
flightHours	☐	long	☐		
Aircraft_aircraft_model	☐	String	✓		14

Aggregate (Output)

Column	K...	Type	N.	Date ...
AIMSyear	☐	String	✓	
Aircraft_aircraft_model	☐	String	✓	
totalFH	☐	Long	✓	

OK Cancel

(10)We then finally configure the aggregation component by specifying the **Group by** columns and the aggregation **Operation** (on which input column to be applied – **flightHours**, function – **sum**, and which output column to fill in with the result – **totalFH**). See figure below.



tAggregateRow_1

Basic settings

Group by

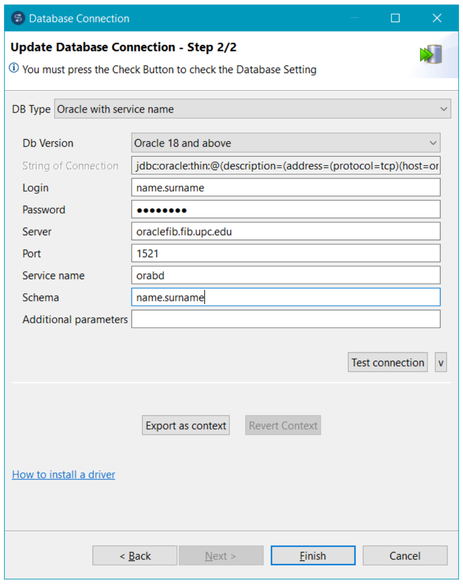
Output column	Input column position
Aircraft_aircraft_model	Aircraft_aircraft_model
AIMSyear	AIMSyear

Operations

Output column	Function	Input column position	Ignore null values
totalFH	sum	flightHours	☐

(11)Finally, after we finish the data flow, we complete it by loading the data resulting from the data flow to the output DW table. In this case, our DW is in Oracle database, so we need to add **tOracleOutput** component from the **Palette** (**Databases** -> **DB Specifics** -> **Oracle** -> **tOracleOutput**). To do this, we need to previously add the connection to our Oracle database to the Metadata repository (See the steps in

Defining DB connection metadata section and parameters in the figure below). Like in the previous case, first time you add an oracle connection, you may be asked to install the corresponding JDBC driver.


Database Connection

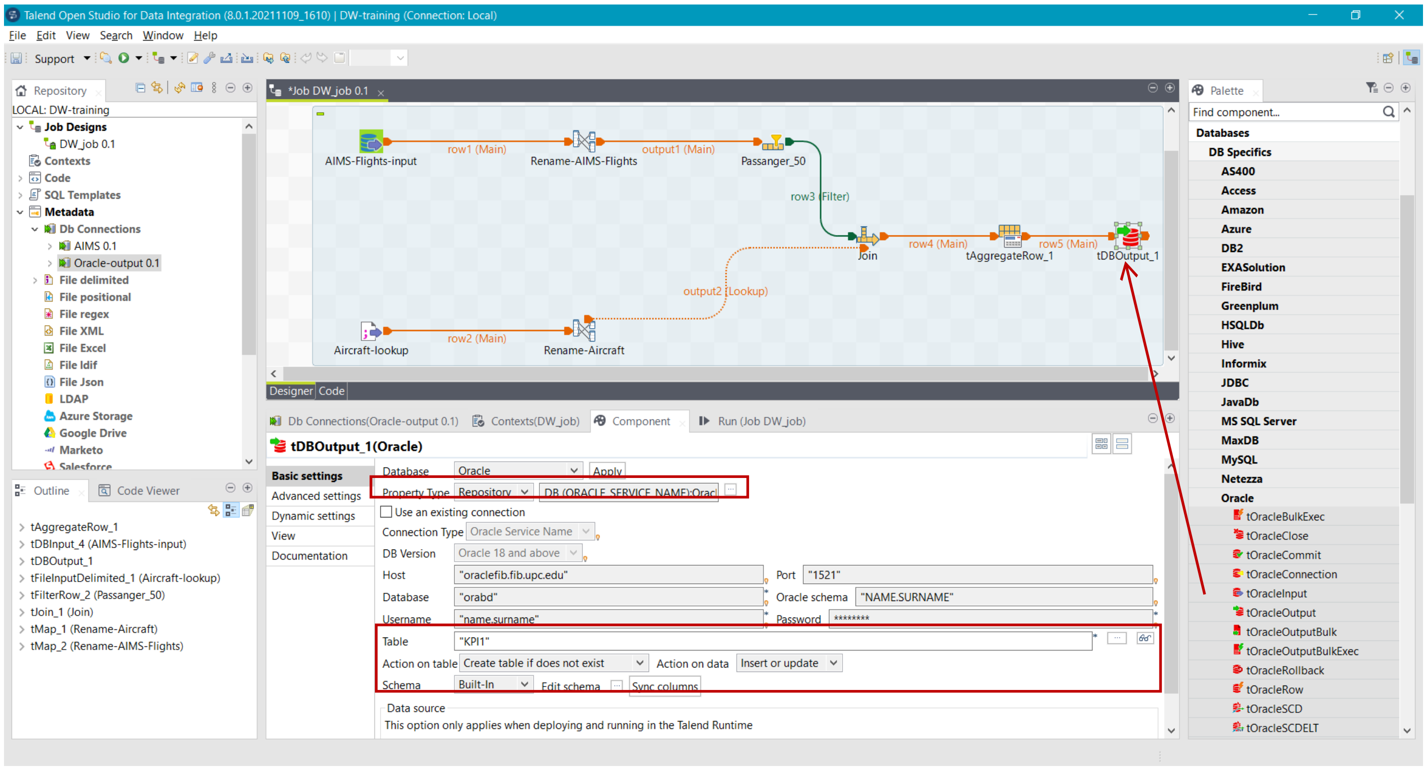
Update Database Connection - Step 2/2

You must press the Check Button to check the Database Setting

DB Type: Oracle with service name

Db Version	Oracle 18 and above
String of Connection	jdbc:oracle:thin:@(description=(address=(protocol=tcp)(host=or
Login	name.surname
Password	••••••••
Server	oraclefib.fib.upc.edu
Port	1521
Service name	orabd
Schema	name.surname
Additional parameters	

We then configure the **tOracleOutput** component as in the following figure. Notice that we choose the previously created connection from the **Repository** as the **Property Type**.

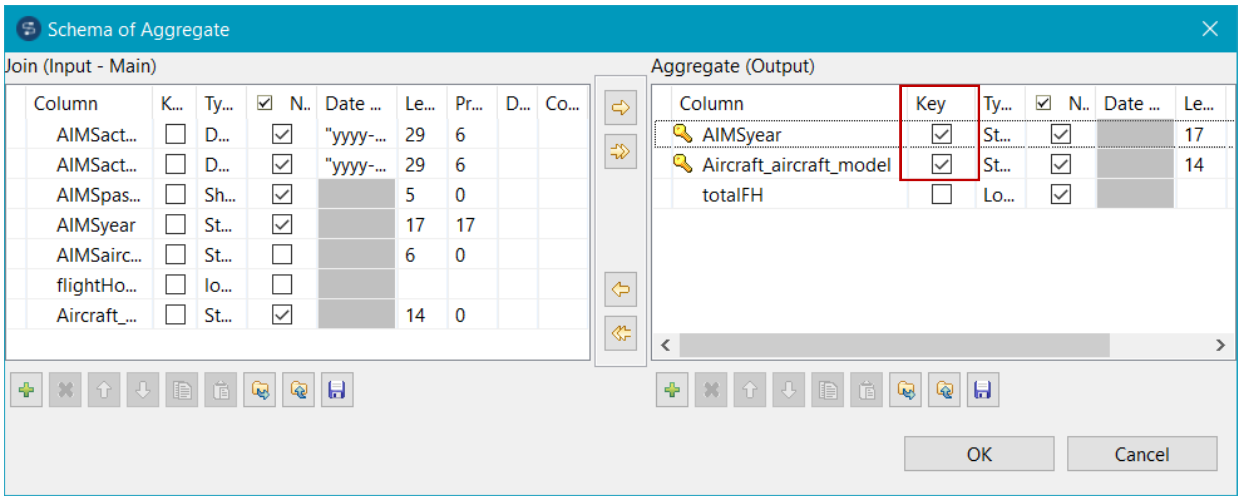


tDBOutput_1(Oracle)

Database	Oracle		
Property Type	Repository	DB (ORACLE_SERVICE_NAME):Oracl	
[ ] Use an existing connection			
Connection Type	Oracle Service Name		
DB Version	Oracle 18 and above		
Host	"oraclefib.fib.upc.edu"	Port	"1521"
Database	"orabd"	Oracle schema	"NAME.SURNAME"
Username	"name.surname"	Password	********
Table	"KPI1"		
Action on table	Create table if does not exist	Action on data	Insert or update
Schema	Built-In		

Data source
This option only applies when deploying and running in the Talend Runtime

We add the name of the output table and we select the actions **Create table if does not exists** and **Insert or update**, to (re)create a KPI1 table where data will be stored and to insert new data or update existing data in the KPI1 table. Notice that for the option **Insert or update**, we need to have clearly defined keys of the table we are inserting. This can be done either directly in the database by setting a PRIMARY KEY, or in the same data flow by editing the output schema of the previous aggregate step (**Edit schema …**) and setting the **Key** property for the group by columns. See figure below.


Schema of Aggregate

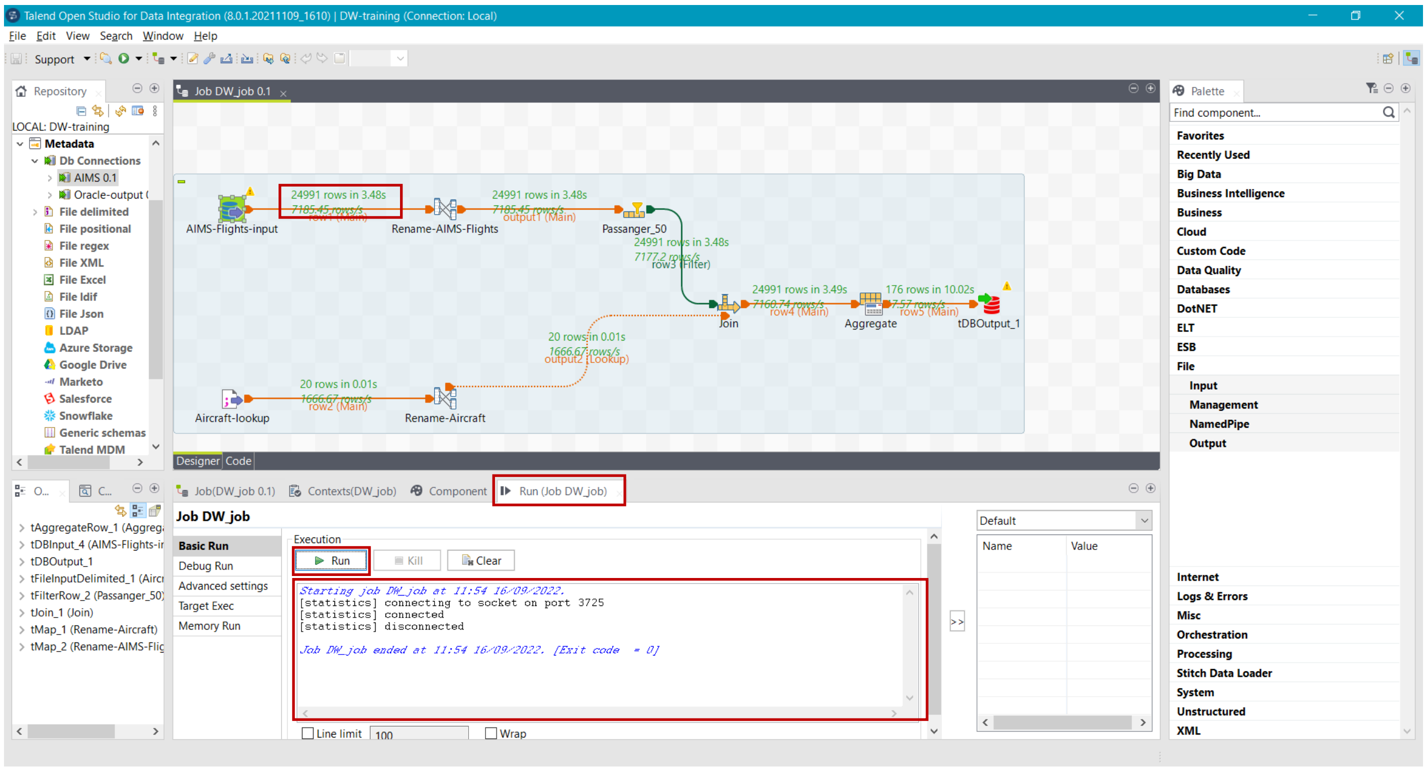
Join (Input - Main)

Column	K...	Ty...	☑ N..	Date ...	Le...	Pr...	D...	Co...
AIMSact...	☐	D...	☑	"yyyy-...	29	6		
AIMSact...	☐	D...	☑	"yyyy-...	29	6		
AIMSpas...	☐	Sh...	☑		5	0		
AIMSyear	☐	St...	☑		17	17		
AIMSairc...	☐	St...	☐		6	0		
flightHo...	☐	lo...	☐					
Aircraft_...	☐	St...	☑		14	0		

Aggregate (Output)

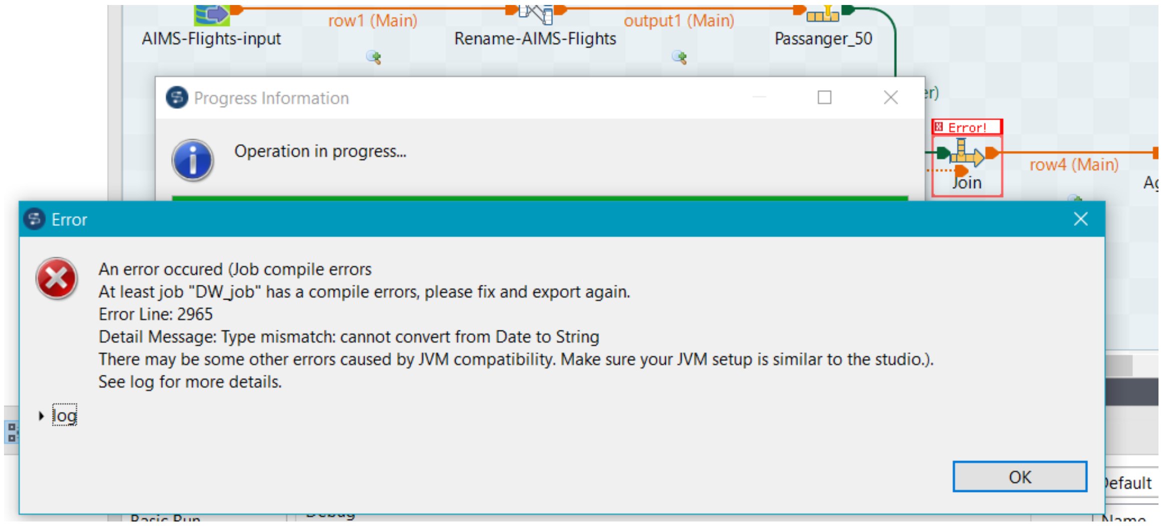
Column	Key	Ty...	☑ N..	Date ...	Le...
AIMSyear	☑	St...	☑		17
Aircraft_aircraft_model	☑	St...	☑		14
totalFH	☐	Lo...	☑		

(12)After we create the ETL process that answers the given KPI 1 we can run the flow by opening the **Run** tab of the control panel. Under the **Basic Run** section, we can just click **Run** and wait for the job to finish. During the execution we will see some quick statistics about the number of rows and latency of the flow in each of the connection. In addition, below we will also find a log that will inform about the flow execution and possible error that may occur. In this case, there are no errors and the job finishes with success (i.e., Exit code = 0)



```
Starting job DW_job at 11:54 16/09/2022.
[statistics] connecting to socket on port 3725
[statistics] connected
[statistics] disconnected

Job DW_job ended at 11:54 16/09/2022. [Exit code  = 0]
```

If there are any **compile time errors** in the created ETL flow, upon running, you should get an error message indicating what could be the issue, while the component where the error occurred will be circled in red with Error! message above the icon, like in the figure below (e.g., because of the type mismatch).



An error occured (Job compile errors
At least job "DW_job" has a compile errors, please fix and export again.
Error Line: 2965
Detail Message: Type mismatch: cannot convert from Date to String
There may be some other errors caused by JVM compatibility. Make sure your JVM setup is similar to the studio.).
See log for more details.

However, an error can also occur at **runtime,** for instance, due to faulty database connection or because of bad data at input.

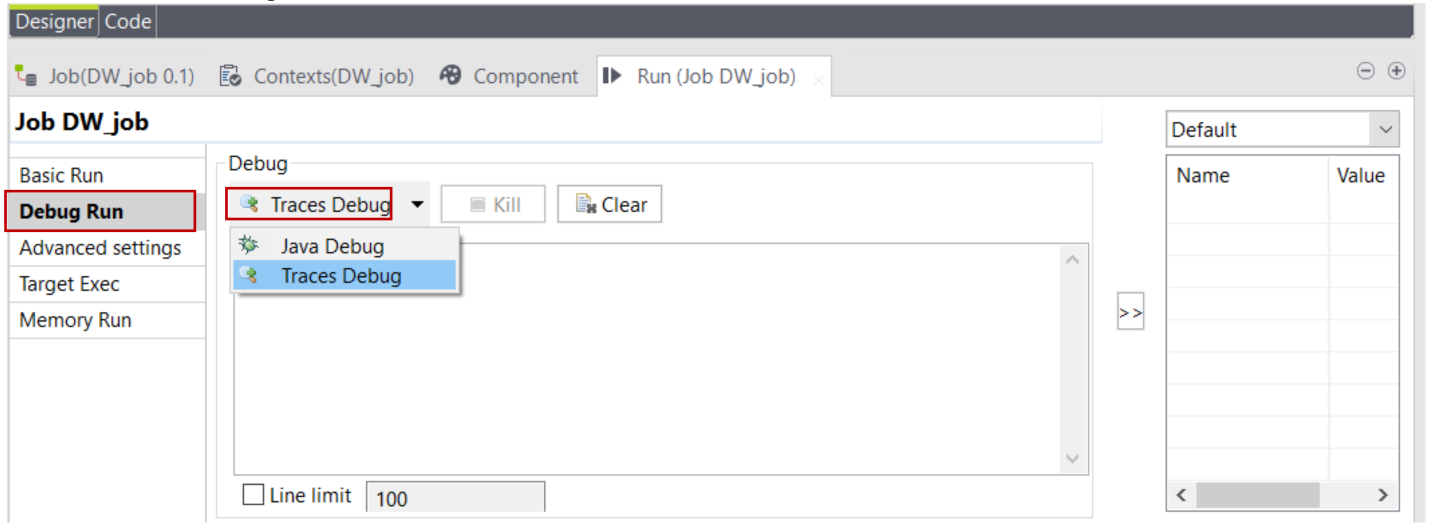
Debugging runtime errors in Talend

In the case, some error occur in the execution (e.g., due to the faulty data like NULL values), Talend provides tools to conveniently debug your ETL flow – The **Traces Debug** mode.

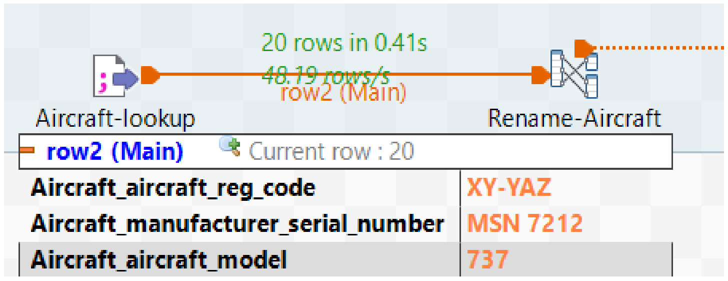
In the Traces Debug mode, Talend Studio provides the capability to monitor the data row-by-row as it flows between different components in an ETL flow. You can choose to **activate** or **deactivate** Traces or decide what processed columns to display in the traces table that displays on the design workspace when launching the current Job. You can either choose to monitor the whole data processing or monitor the data processing row-by-row or at a certain breakpoint.

To debug the Job in the Traces Debug mode:

1. With your Job open in Talend Studio, open the **Run** tab in the control panel and then select **Debug Run**.
2. Click the **Traces Debug** button[9].



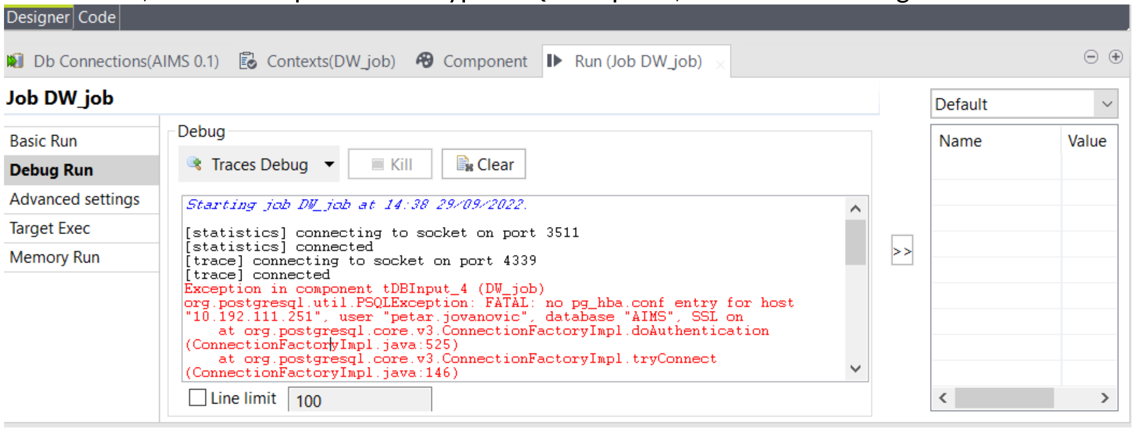
The Job runs, with row content displayed under each main output of a component, like it is shown in the figure below.



Aircraft_aircraft_reg_code	XY-YAZ
Aircraft_manufacturer_serial_number	MSN 7212
Aircraft_aircraft_model	737

You can watch the content of the rows as they flow through the ETL flow until it encounters an error.

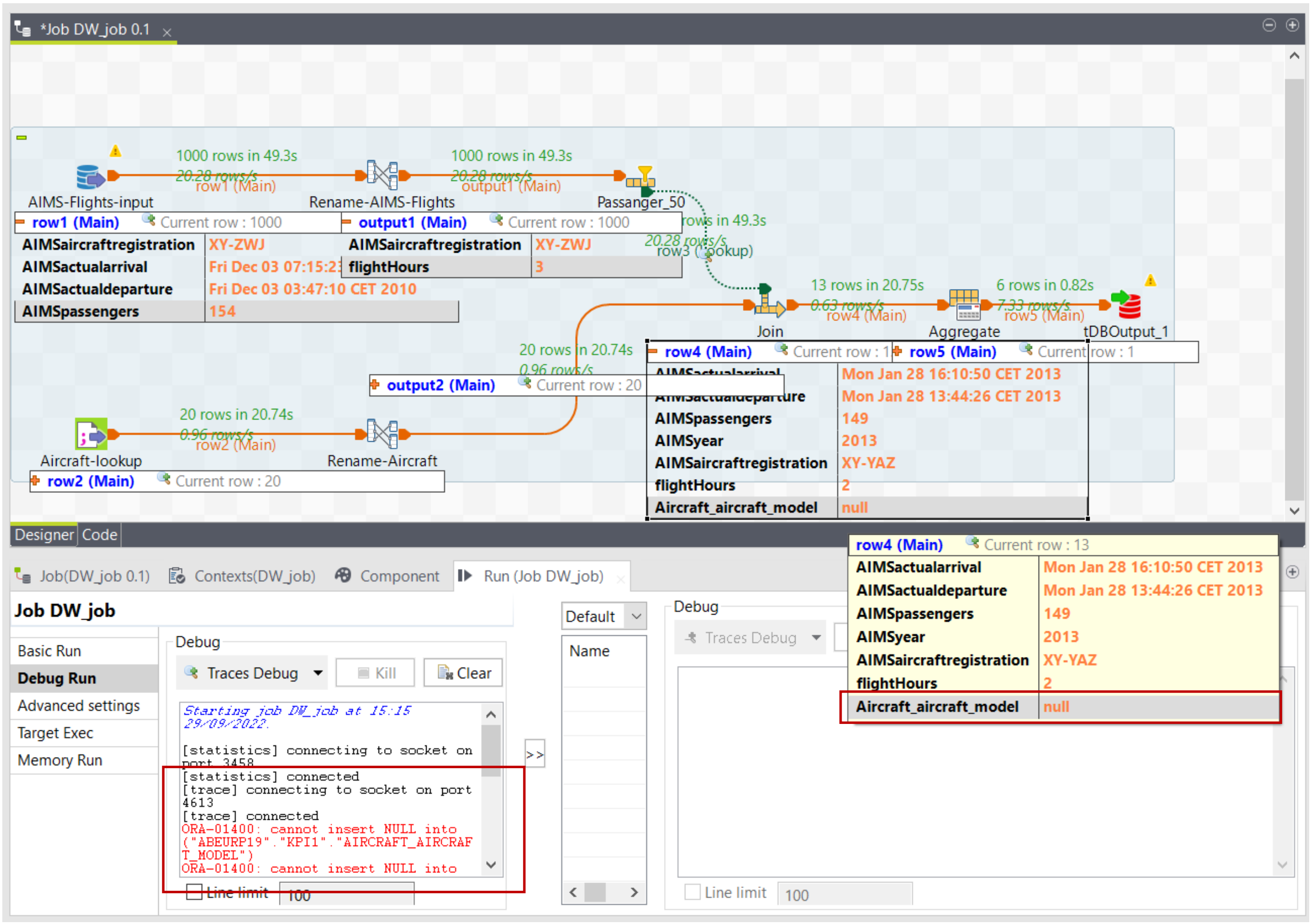
3. If an error occurs because of a not properly configured database connection, or if you are not connected to the VPN of UPC, the error reported is of type PSQLException, as shown in the figure below



```
Starting job DW_job at 14:38 29/09/2022.

[statistics] connecting to socket on port 3511
[statistics] connected
[trace] connecting to socket on port 4339
[trace] connected
Exception in component tDBInput_4 (DW_job)
org.postgresql.util.PSQLException: FATAL: no pg_hba.conf entry for host
"10.192.111.251", user "petar.jovanovic", database "AIMS", SSL on
    at org.postgresql.core.v3.ConnectionFactoryImpl.doAuthentication
(ConnectionFactoryImpl.java:525)
    at org.postgresql.core.v3.ConnectionFactoryImpl.tryConnect
(ConnectionFactoryImpl.java:146)
```

4. However, if an error occurs because of bad data (which is more often), you can examine the values of each field in the row that caused the error. In this case, the error reported is a null pointer exception or error when inserting a NULL value to an output database table, as shown below.

[9] Note: Click the triangle in the right part of the button and select Traces Debug if Traces Debug does not appear on the button.



row1 (Main)	Current row : 1000
AIMSaircraftregistration	XY-ZWJ
AIMSactualarrival	Fri Dec 03 07:15:2
AIMSactualdeparture	Fri Dec 03 03:47:10 CET 2010
AIMSpassengers	154

output1 (Main)	Current row : 1000
AIMSaircraftregistration	XY-ZWJ
flightHours	3

row4 (Main)	Current row : 13
AIMSactualarrival	Mon Jan 28 16:10:50 CET 2013
AIMSactualdeparture	Mon Jan 28 13:44:26 CET 2013
AIMSpassengers	149
AIMSyear	2013
AIMSaircraftregistration	XY-YAZ
flightHours	2
Aircraft_aircraft_model	null

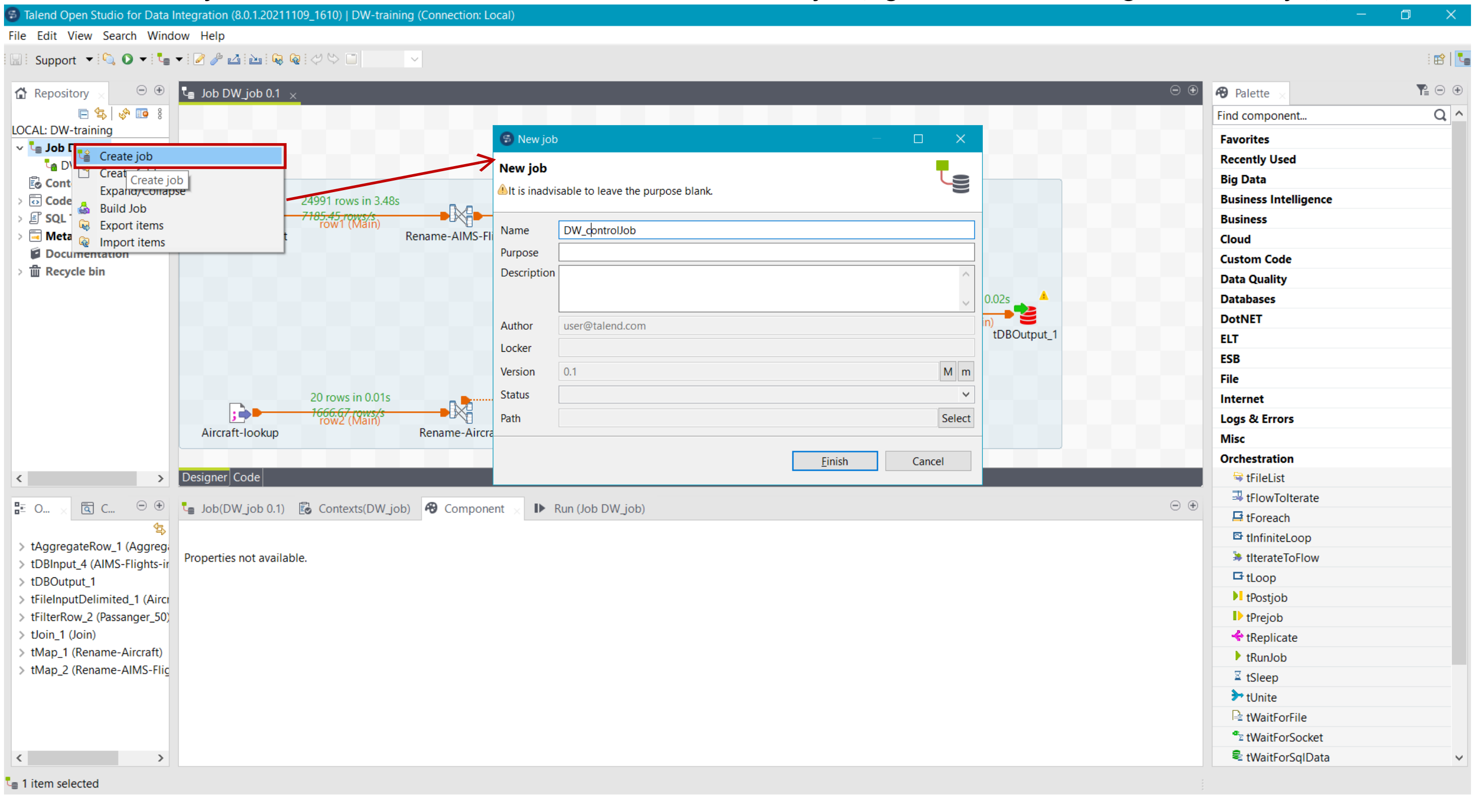
```
Starting job DW_job at 15:15
29/09/2022.

[statistics] connecting to socket on
port 3458
[statistics] connected
[trace] connecting to socket on port
4613
[trace] connected
ORA-01400: cannot insert NULL into
("ABEURP19"."KPI1"."AIRCRAFT_AIRCRAF
T_MODEL")
ORA-01400: cannot insert NULL into
```

This is an excellent way of debugging Jobs when you are working with a relatively small sample of data. However, debugging Jobs that handle large volumes of data in this mode is not recommended because displaying every row visually in the Studio console can slow down the execution of the Job. In this case, you can try the Java Debug mode[10].

[10] https://help.talend.com/r/en-US/8.0/debug-jobs/java-debug-mode

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

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



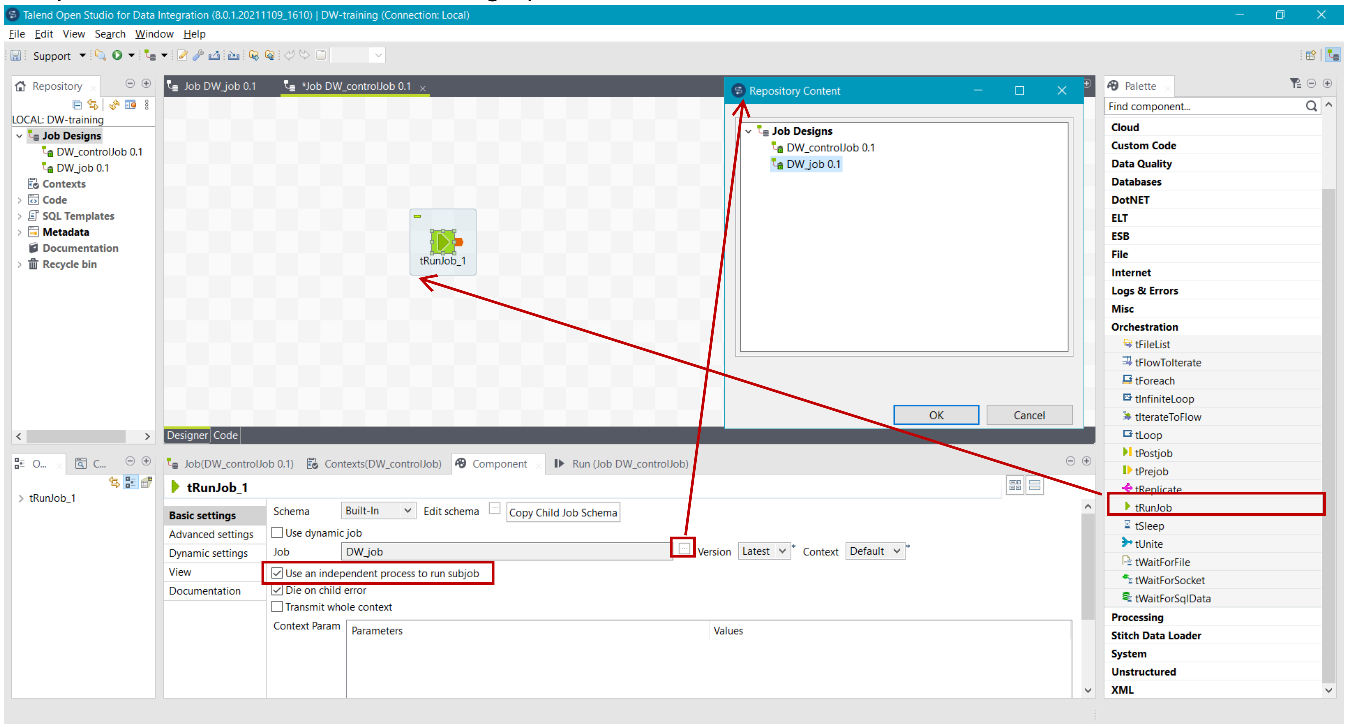
New job

It is inadvisable to leave the purpose blank.

Name	DW_controlJob
Purpose	
Description	
Author	user@talend.com
Locker	
Version	0.1
Status	
Path	

Properties not available.

We next need to connect the control flow job to the previously created data flow job. To do this, we use the **RunJob** component from the **Orchestration** category of the **Palette.**



tRunJob_1

Basic settings

Schema: Built-In — Edit schema — Copy Child Job Schema
Use dynamic job
Job: DW_job — Version: Latest — Context: Default
☑ Use an independent process to run subjob
☑ Die on child error
Transmit whole context
Context Param: Parameters / Values

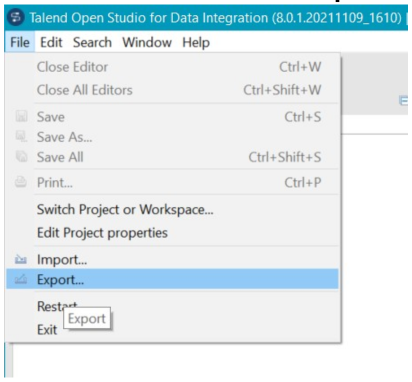
We first need to link the **RunJob** component to the data flow job by selecting it from the project repository (**DW_job**). In addition, we can also configure that the **RunJob** component uses an independent process to run the given data flow (**subjob**), by selecting this option in the component configuration. See the figure above.
We run the control flow job in the same manner as the data flow job, i.e., by using the **Run** tab in the control panel. Similarly, we can add other data flows as subjobs using the **RunJob** component and connect them in order to orchestrate their execution, or use other orchestration components[11]

[11] https://help.talend.com/r/en-US/8.0/orchestration/orchestration (Notice that not all the orchestrating components are available in the open source version of Talend).

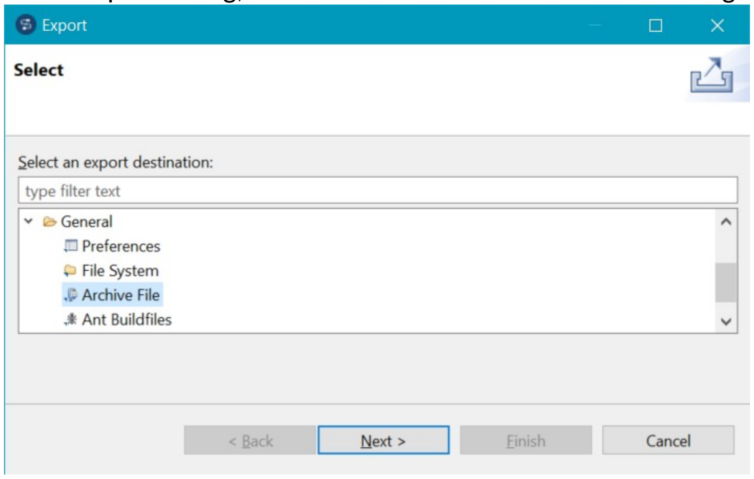
Appendix A - Exporting Talend project

To export your Talend project, in order to use it in another machine or for the final project delivery, you can follow these steps.

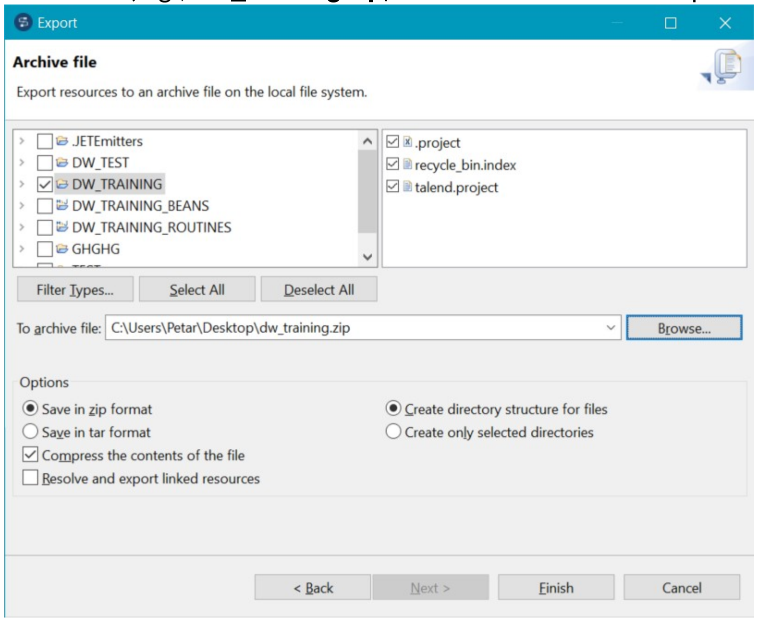
1. Under **File** menu click on **Export**


Talend Open Studio for Data Integration (8.0.1.20211109_1610)
File Edit Search Window Help
Close Editor Ctrl+W
Close All Editors Ctrl+Shift+W
Save Ctrl+S
Save As...
Save All Ctrl+Shift+S
Print... Ctrl+P
Switch Project or Workspace...
Edit Project properties
Import...
Export...
Restart
Exit
Export

2. In the Export dialog, choose **Archive File** under **General** category, as a type of the export artifact. Then click Next.



3. Select the Talend project you want to export, choose the destination folder (**Browse...**) and give it a name with the extension (e.g., **dw_training.zip**). Be sure that the other options are as in the figure below. Then click Finish.



- [ ] .JETEmitters
- [ ] DW_TEST
- [x] DW_TRAINING
- [ ] DW_TRAINING_BEANS
- [ ] DW_TRAINING_ROUTINES
- [ ] GHGHG

- [x] .project
- [x] recycle_bin.index
- [x] talend.project

To archive file: C:\Users\Petar\Desktop\dw_training.zip

Options

[✓ Save in zip format] Save in tar format

[✓ Create directory structure for files] Create only selected directories

- [x] Compress the contents of the file
- [ ] Resolve and export linked resources

Your project is exported in a .zip archive file, and as such can be imported into another Talend installation[12]

[12] Import project: https://help.talend.com/r/en-US/8.0/studio-user-guide-open-studio-for-data-integration/importing-local-projects