Creating Date Dimensions with Matillion ETL

You may have stumbled across this article looking for help creating or modifying an existing date/time/calendar dimension. Or you may be struggling with dates in your reports or analytical tools. Everyone needs to set up a date dimension. This article will help you understand how to easily create a date dimension within Matillion ETL. For the sake of this article, I will be referring to the date dimension as a calendar dimension to avoid confusion.

Why use Matillion ETL to create dimension tables?

Matillion ETL enables you to migrate and load data into a cloud data warehouse such as Amazon Redshift, Google BigQuery, Snowflake, or Azure Synapse Analytics. You can then transform that data into a suitable format for analytics. This is all done within your cloud data warehouse (CDW) environment. It is worth mentioning that Matillion ETL instances can run on multiple cloud platforms: Amazon Web Services(AWS), Microsoft Azure and Google Cloud Platform(GCP), but keep in mind that some CDWs are only offered on certain cloud providers. For example, Snowflake runs on AWS, Azure and GCP, however, Amazon Redshift only runs on AWS and Google BigQuery only runs on GCP. As part of that data transformation process,

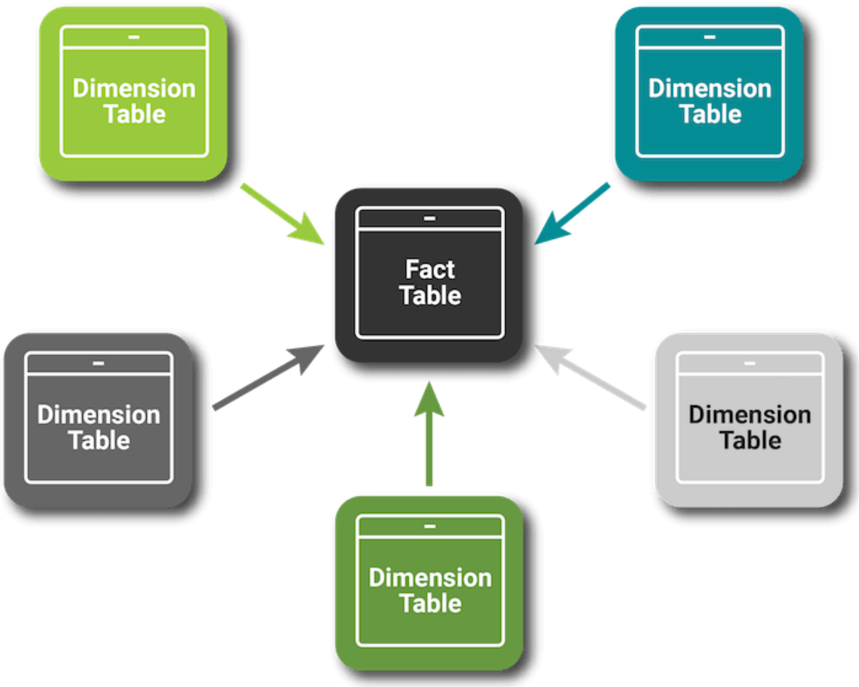
Creating Date Dimensions with Matillion ETL

Matillion ETL creates dimension tables that can be used in future Transformation Jobs, which helps guarantee that the data going in for analytics will be transformed, well prepared, and yield optimal results.

What is a dimension?

Before moving onto calendar dimensions, let's step back a little bit. What is a dimension?

A dimension is a **collection of reference information about a measurable event**). Dimensionality is a key aspect of data warehousing. You can deliver dimensionality through a combination of "measurements" and the "context" for those measurements. The measurements in this case are called "facts" and the context is "dimensions".



Creating Date Dimensions with Matillion ETL

Fact tables and dimension tables

When we use a relational database as the platform for our data warehouse, we store our 'facts' in a 'fact table'. Likewise, we store our dimensions in a dimension table. A dimension table is *usually* accompanied by a **surrogate key**. The surrogate key is used to relate data across the tables and is usually generated by the database, which acts both as a primary and a foreign key in fact tables.

The fact and dimension tables have a granularity associated with them. In dimensional modelling, granularity refers to the level of detail stored in a table. For example, a calendar dimension table (with Day, Month and Year hierarchies) has a granularity at the day level (365 rows for a year).

Why create calendar dimensions?

Calendar dimensions are useful in a lot of cases. A calendar dimension contains all of the information about a certain date and allows analysts and users to analyse data as efficiently and accurately as possible. This information is then at your disposal eliminating the need to perform unnecessary calculations. Usually a calendar dimension stores historical data, dates that go back to as far as the early 1900s or as far as needed for the facts. But future dates can be used as well. Some common attributes that accompany a date dimension are: isHoliday, DayofWeek, Quarter.

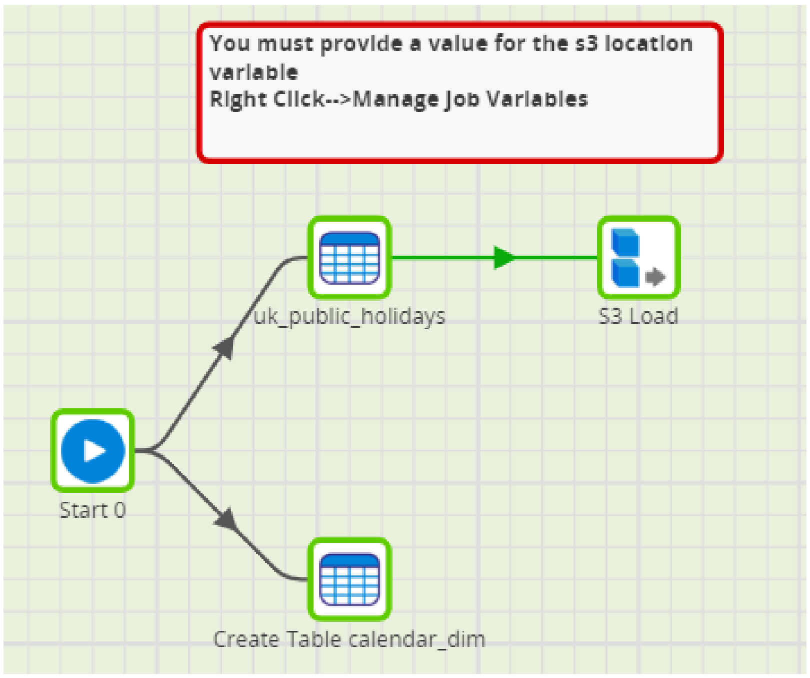
Building a Calendar Dimension

Using a Redshift on AWS instance, we used Matillion ETL to ingest the data by creating an Orchestration Job. An Orchestration Job consists of several components that can retrieve data from other services and load that data into a table. For example, these components can be called Data Stagers, Connectors, or Data Loaders.

Creating Date Dimensions with Matillion ETL

Loading the data

First, I created an empty table that will hold the calendar dimension and a staging table, where the extracted data will sit. To connect to this table, I used an S3 Load component, which extracts the data from a CSV file stored in an S3 bucket and automatically loads it into the staging table.The initial data came from **https://www.ukbankholidays.co.uk/year/2000** and I compiled it into a single CSV file that you can download **here**.

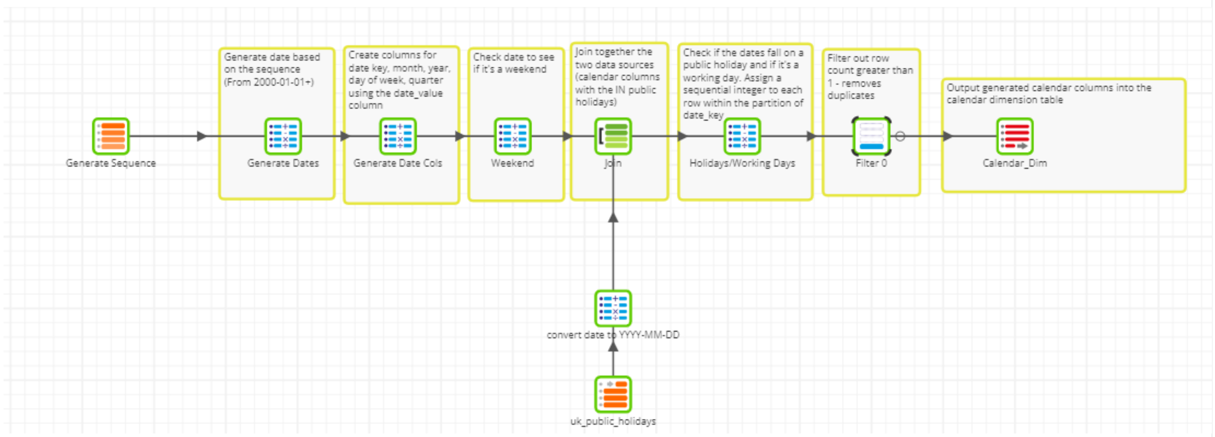

You must provide a value for the s3 location variable
Right Click-->Manage Job Variables

Data Transformation in a calendar dimension

I can now start generating the necessary attributes for our calendar dimension table.

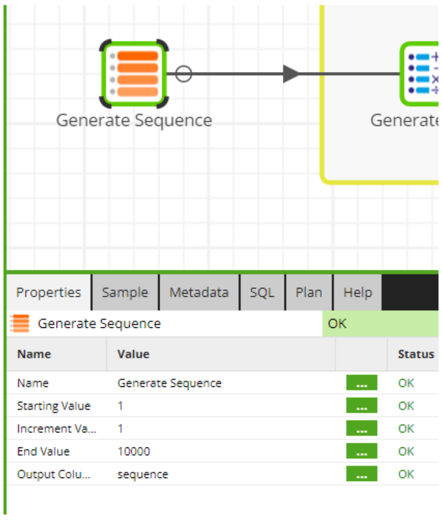
I created a Transformation Job, which will help me generate the necessary columns to shape up the calendar dimension. For ease of future maintenance, we recommend not to hand-code jobs with SQL. Instead, components provide the necessary blocks of code. This is what the full Transformation Job looks like:

Creating Date Dimensions with Matillion ETL


Generate Sequence
Generate date based on the sequence (From 2000-01-01+)
Generate Dates
Create columns for date key, month, year, day of week, quarter using the date_value column
Generate Date Cols
Check date to see if it's a weekend
Weekend
Join together the two data sources (calendar columns with the IN public holidays)
Join
Check if the dates fall on a public holiday and if it's a working day. Assign a sequential integer to each row within the partition of date_key
Holidays/Working Days
Filter out row count greater than 1 - removes duplicates
Filter 0
Output generated calendar columns into the calendar dimension table
Calendar_Dim
convert date to YYYY-MM-DD
uk_public_holidays

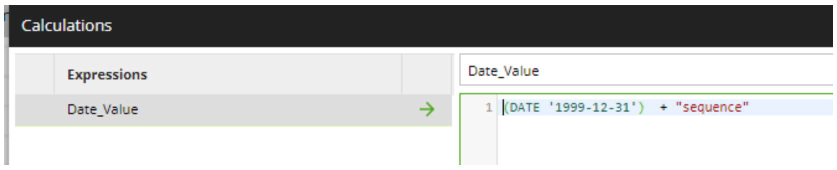
How do each of these components give us what we need? Let's take a look.

The Generate Sequence Component



Generate Sequence OK

Name	Value		Status
Name	Generate Sequence	...	OK
Starting Value	1	...	OK
Increment Va...	1	...	OK
End Value	10000	...	OK
Output Colu...	sequence	...	OK



Expressions
Date_Value

Date_Value

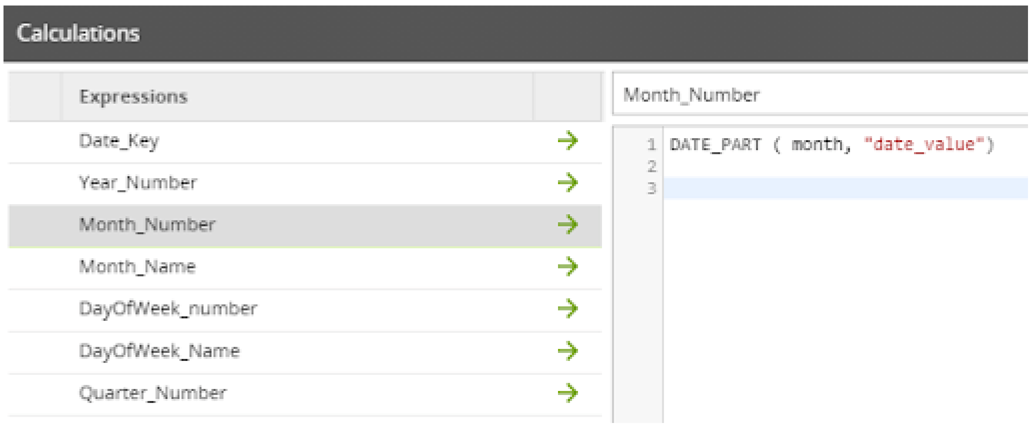
```
(DATE '1999-12-31')  + "sequence"
```

Creating Date Dimensions with Matillion ETL

This component in itself simply generates a sequence from 1 to 10000. I then used the sequence generated to generate the dates from 2000-01-01 and onwards. It's a simple process and saves a lot of time.

The Calculator Component

Using the Calculator component, I created columns for the date key, month, year, day of the week and quarter number using the 'date_value' that we generated in the previous step. A benefit of the Calculator component is that you do not need to remember both the function names and syntax. All of that information is provided under the functions folder. The functions available will be native to the data warehouse logic.



```
DATE_PART ( month, "date_value")
```

Now we can check to see if the date falls on a weekend.

We do this by using a CASE statement.

CASE

WHEN "dayofweek_number" IN (0,6) THEN 1

ELSE 0

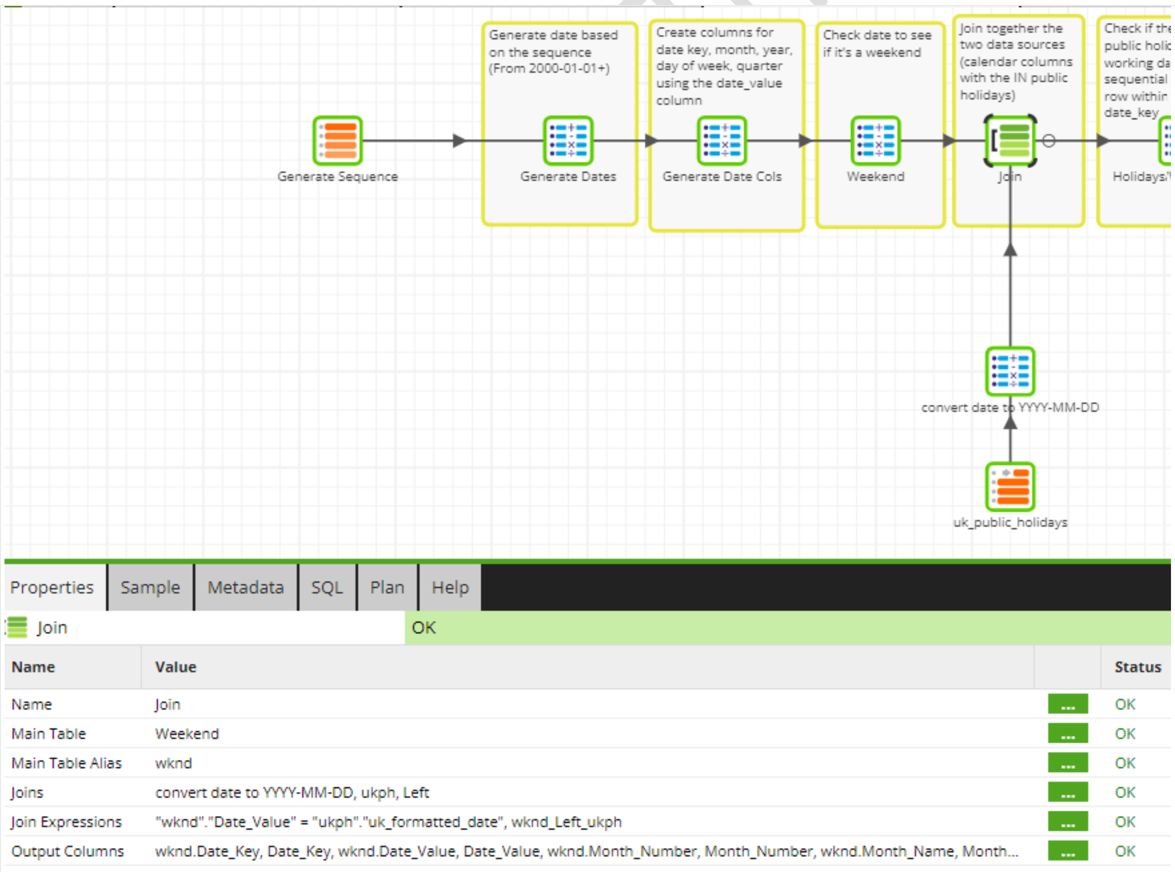
END

Creating Date Dimensions with Matillion ETL

The Join Component

Next, we need to join two data sources (current calendar columns and UK Public Holidays) to aid in generating the two final columns. The Join component joins the columns on a column key. Since the two tables have a matching key (date_value), I used a left join to receive data from the left table, even if there are no matches in the right table. You can also choose between the following join types:

- Inner
- Right
- Full



Generate date based on the sequence (From 2000-01-01+)

Create columns for date key, month, year, day of week, quarter using the date_value column

Check date to see if it's a weekend

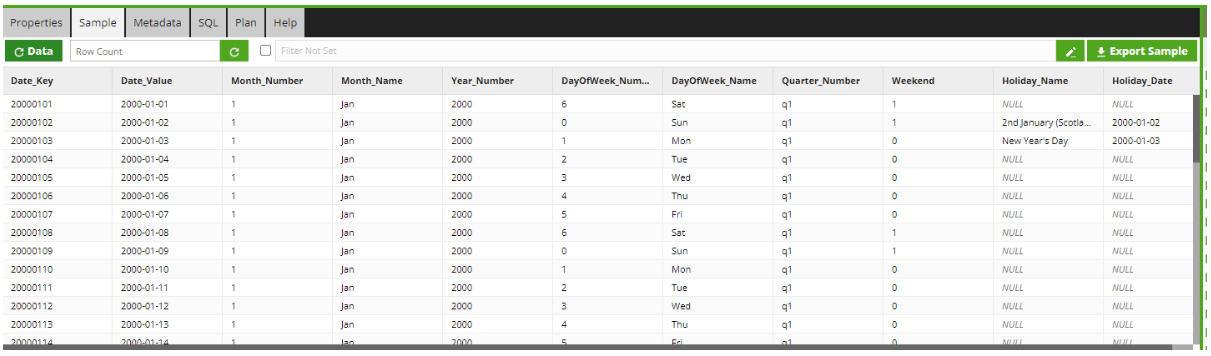
Join together the two data sources (calendar columns with the IN public holidays)

Check if the public holic working da sequential row within date_key

Join OK

Name	Value		Status
Name	Join	...	OK
Main Table	Weekend	...	OK
Main Table Alias	wknd	...	OK
Joins	convert date to YYYY-MM-DD, ukph, Left	...	OK
Join Expressions	"wknd"."Date_Value" = "ukph"."uk_formatted_date", wknd_Left_ukph	...	OK
Output Columns	wknd.Date_Key, Date_Key, wknd.Date_Value, Date_Value, wknd.Month_Number, Month_Number, wknd.Month_Name, Month...	...	OK

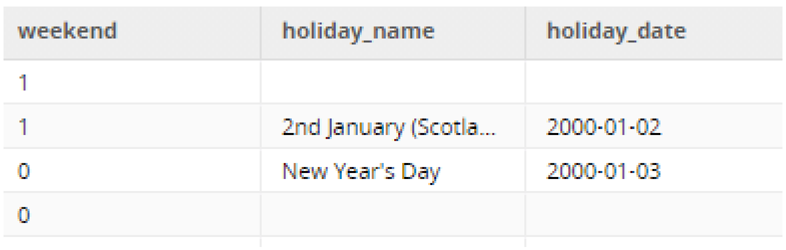
Creating Date Dimensions with Matillion ETL



Properties | Sample | Metadata | SQL | Plan | Help

Data | Row Count | Filter Not Set | Export Sample

Date_Key	Date_Value	Month_Number	Month_Name	Year_Number	DayOfWeek_Num...	DayOfWeek_Name	Quarter_Number	Weekend	Holiday_Name	Holiday_Date
20000101	2000-01-01	1	Jan	2000	6	Sat	q1	1	NULL	NULL
20000102	2000-01-02	1	Jan	2000	0	Sun	q1	1	2nd January (Scotla...	2000-01-02
20000103	2000-01-03	1	Jan	2000	1	Mon	q1	0	New Year's Day	2000-01-03
20000104	2000-01-04	1	Jan	2000	2	Tue	q1	0	NULL	NULL
20000105	2000-01-05	1	Jan	2000	3	Wed	q1	0	NULL	NULL
20000106	2000-01-06	1	Jan	2000	4	Thu	q1	0	NULL	NULL
20000107	2000-01-07	1	Jan	2000	5	Fri	q1	0	NULL	NULL
20000108	2000-01-08	1	Jan	2000	6	Sat	q1	1	NULL	NULL
20000109	2000-01-09	1	Jan	2000	0	Sun	q1	1	NULL	NULL
20000110	2000-01-10	1	Jan	2000	1	Mon	q1	0	NULL	NULL
20000111	2000-01-11	1	Jan	2000	2	Tue	q1	0	NULL	NULL
20000112	2000-01-12	1	Jan	2000	3	Wed	q1	0	NULL	NULL
20000113	2000-01-13	1	Jan	2000	4	Thu	q1	0	NULL	NULL
20000114	2000-01-14	1	Jan	2000	5	Fri	q1	0	NULL	NULL



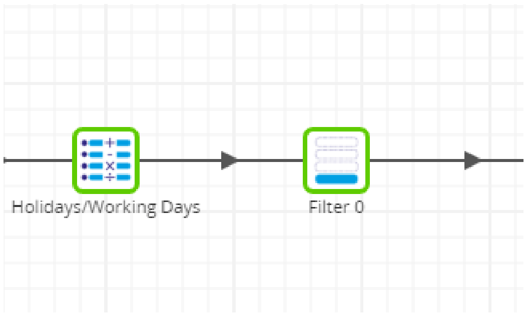
weekend	holiday_name	holiday_date
1		
1	2nd January (Scotla...	2000-01-02
0	New Year's Day	2000-01-03
0		

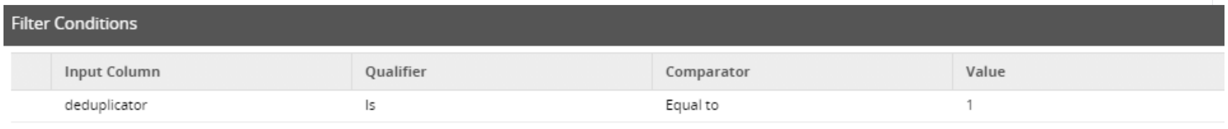
Another Calculator component creates two columns, which will check to see which dates fall on a holiday or a working day, taking into account weekends and public holidays. There is also the matter of two public holidays falling on the same day, creating a few duplicate 'date_key' rows. To work around this, assign a sequential integer to each row within the partition of the date_key:

ROW_NUMBER() OVER (PARTITION BY "date_key")

The Filter Component

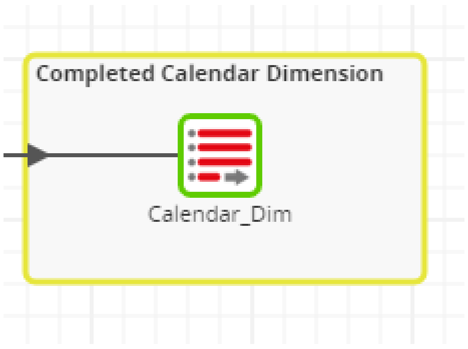
In order to get rid of the duplicates, I created a field that I named 'deduplicator' and used a Filter Component to filter out values that are greater than 1. This is done within a simple window, again portraying how easy it is to pull outcomes without the need to hand-code anything.




Filter Conditions

	Input Column	Qualifier	Comparator	Value
	deduplicator	Is	Equal to	1

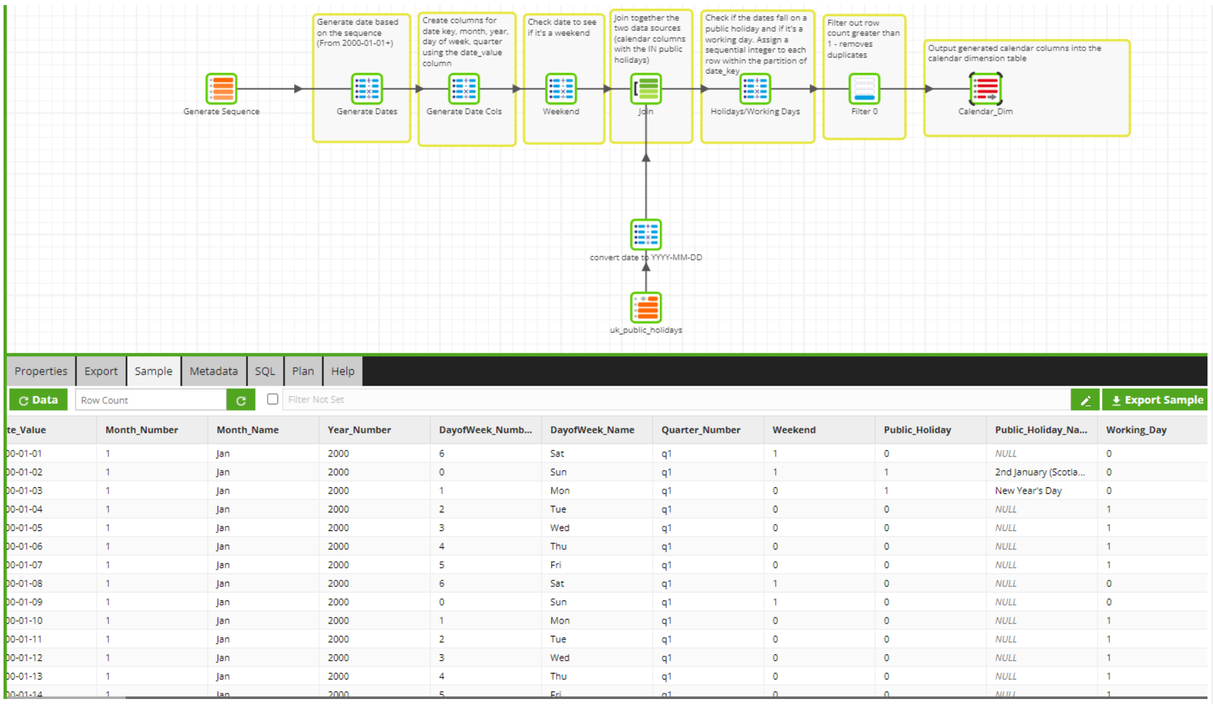
The Table Output Component



Right at the start we created an empty table in the Orchestration Job. Now, I can use the Table Output Component to push the data into the 'calendar_dim' table. In doing so, I will make sure that the Truncate property is set to Truncate, not Append, so that it does not create duplicate rows.

ANALYTICS WITH ANAND
LET THE DATA DO THE TALKING...

Creating Date Dimensions with Matillion ETL


Generate Sequence
Generate date based on the sequence (From 2000-01-01+)
Generate Dates
Create columns for date key, month, year, day of week, quarter using the date_value column
Generate Date Cols
Check date to see if it's a weekend
Weekend
Join together the two data sources (calendar columns with the IN public holidays)
Join
Check if the dates fall on a public holiday and if it's a working day. Assign a sequential integer to each row within the partition of date_key
Holidays/Working Days
Filter out row count greater than 1 - removes duplicates
Filter 0
Output generated calendar columns into the calendar dimension table
Calendar_Dim
convert date to YYYY-MM-DD
uk_public_holidays
Properties
Export
Sample
Metadata
SQL
Plan
Help
Data
Row Count
Filter Not Set
Export Sample

te_Value	Month_Number	Month_Name	Year_Number	DayofWeek_Numb...	DayofWeek_Name	Quarter_Number	Weekend	Public_Holiday	Public_Holiday_Na...	Working_Day
00-01-01	1	Jan	2000	6	Sat	q1	1	0	NULL	0
00-01-02	1	Jan	2000	0	Sun	q1	1	1	2nd January (Scotla...	0
00-01-03	1	Jan	2000	1	Mon	q1	0	1	New Year's Day	0
00-01-04	1	Jan	2000	2	Tue	q1	0	0	NULL	1
00-01-05	1	Jan	2000	3	Wed	q1	0	0	NULL	1
00-01-06	1	Jan	2000	4	Thu	q1	0	0	NULL	1
00-01-07	1	Jan	2000	5	Fri	q1	0	0	NULL	1
00-01-08	1	Jan	2000	6	Sat	q1	1	0	NULL	0
00-01-09	1	Jan	2000	0	Sun	q1	1	0	NULL	0
00-01-10	1	Jan	2000	1	Mon	q1	0	0	NULL	1
00-01-11	1	Jan	2000	2	Tue	q1	0	0	NULL	1
00-01-12	1	Jan	2000	3	Wed	q1	0	0	NULL	1
00-01-13	1	Jan	2000	4	Thu	q1	0	0	NULL	1
00-01-14	1	Jan	2000	5	Fri	q1	0	0	NULL	1

Try it for yourself

You can now start to build out your star schema. We could find further use for this dimension and start linking it to other dimensions to create fact tables. For example in a college scenario, we can use the date key to reference the calendar dimension and use a student key from its respective dimension to build a tuition payment fact table.