Using the From Variable Data route

Revision 20200525

Goal

This document describes how to use the option “`From Variable`” in the data route configuration of a data source.

Content

In Denodo Virtual DataPort, data sources are usually created for extracting data from a known source like a database, a file or any other system containing information. However, at Denodo we have the option of building data sources which are not restricted to a path or a URL. In this case the data source works by accepting the data directly as an input parameter and returning the data extracted in the configured output fields of the base views created on top of a data source created this way.

To create this type of data sources, the “`From Variable`” option is available as a “`Data route`” for **XML**, **JSON** and **Delimited File** data sources. This feature is very useful when we need to combine data in any of these formats and the data is obtained dynamically instead of being stored in a particular folder/file or accessible through a URL. For instance, a CSV file that is composed on the fly or an XML that is contained within a text field of a relational database, in those cases, given the dynamic nature of the source the “`From Variable`” option will be the best way to integrate this kind of data.

Once the raw data is obtained from its original location it can be passed to the “`From Variable`” data source in order to be integrated as needed, this way we can combine it with the other data sources in our Denodo virtual layer.

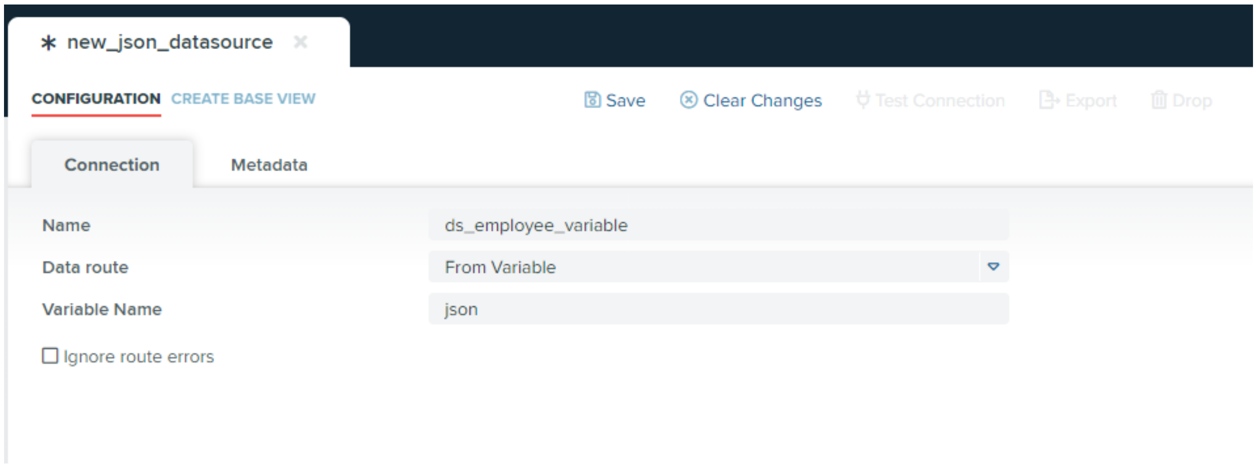
Let’s see a real life example focused on JSON format. Currently, it is very common to store data as JSON files since they are used for a lot of different purposes. JSON is widely used when working with web services and even some of the new NoSQL databases can store and work directly with these files. On the other hand, we have the common relational databases which can be used for persisting JSON fragments as well, because these snippets are plain text after all. The big difference between both systems is that most relational databases do not include specific tools for handling these formats, so a custom client will be required in order to work with them.

For this example, we are going to reuse the JSON created for this Denodo KB document which explains how to access hierarchical data in JSON in Hive. In this environment, each tuple of the Hive table includes a string field containing the JSON that models an employee register. Here it is an example:

```
{ "id" : "1234",
  "name" : "Christopher C Laird",
  "telephone" : "212-618-9176",
  "date" : "04/17/2013",
  "address" : { "city" : "New York",
                "street" : "3204 Oakwood Avenue",
                "zipcode" : "10007"
               },
  "department" : {"name" : "Sales",
                  "director" : "Ellis L Williams",
                  "budget" : "100000",
                  "address" : {"city" : "New York",
                               "street" : "3687 Pallet Street",
                               "zipcode" : "10007"
                              }
                 }
}
```

This time we can choose between two approaches in order to work with this information in Denodo. We can use the custom functions that Hive provides explained in the previous document of the KB or create a From Variable data source. Note that the latter will be mandatory for the sources that do not offer specific JSON functions.

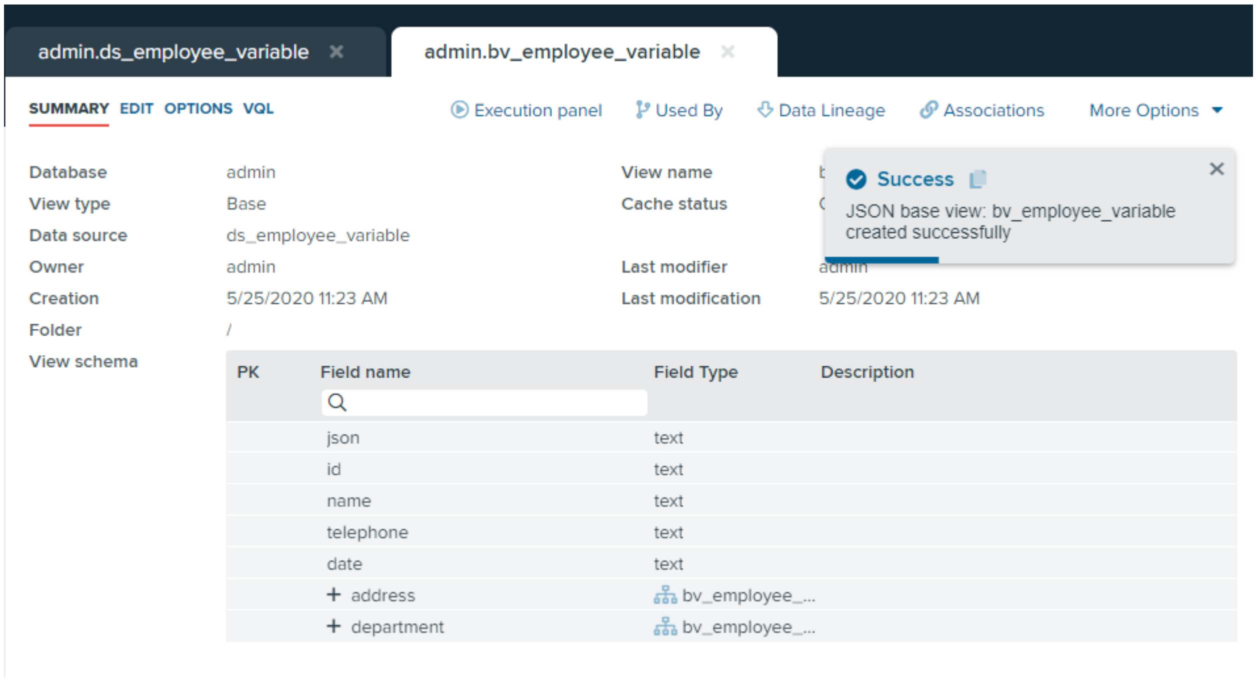
It is very easy to create a JSON data source with the From Variable option, we just need to set “`From Variable`” in the data route configuration and configure the data route setting the variable name, for instance “json”. Let’s call this data source `ds_employee_variable`.



Name: ds_employee_variable
Data route: From Variable
Variable Name: json
☐ Ignore route errors

`ds_employee_variable` data source

Then, for creating a base view over this data source click on Create base view and use the previous JSON document as example, doing this Denodo can introspect the schema of the view automatically. This base view is going to be called `bv_employee_variable`.



Success

JSON base view: bv_employee_variable created successfully

Database	admin	View name	
View type	Base	Cache status	
Data source	ds_employee_variable		
Owner	admin	Last modifier	admin
Creation	5/25/2020 11:23 AM	Last modification	5/25/2020 11:23 AM
Folder	/		

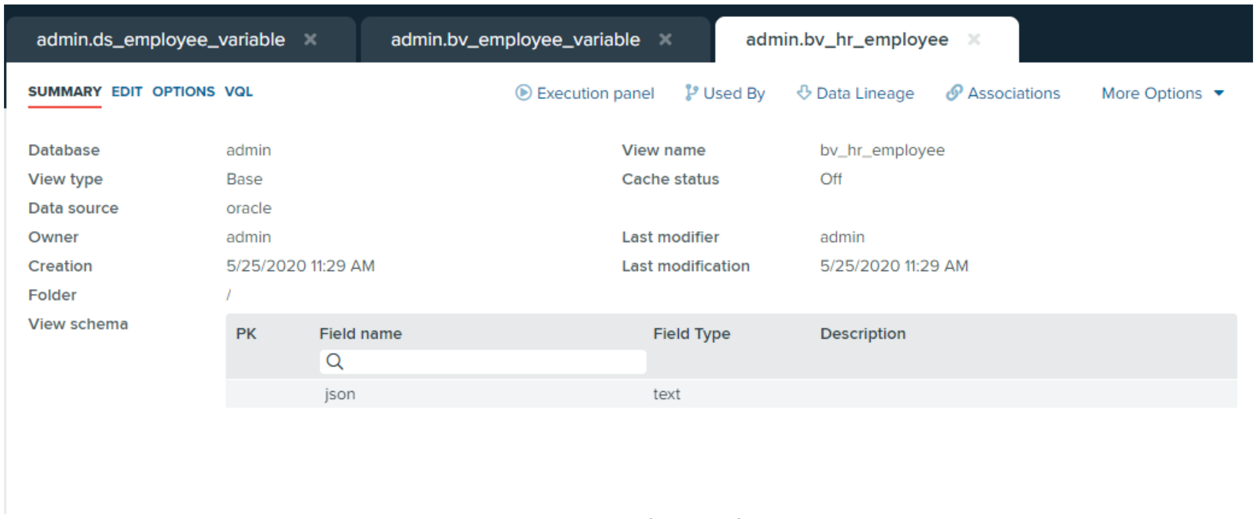
View schema

PK	Field name	Field Type	Description
	json	text	
	id	text	
	name	text	
	telephone	text	
	date	text	
	+ address	bv_employee_...	
	+ department	bv_employee_...	

`bv_employee_variable` base view

If we try to execute the base view we will see that the value for the json mandatory parameter needs to be provided. Later, it will be explained how to fill this parameter automatically.

Likely, the Hive data source that contains the table with the target JSON string is already created in our Virtual Database as a typical JDBC data source. If not, we will need to create it first as any other JDBC data source and second, create the base view, for instance bv_hr_employee, introspecting the JSON table in the schema of the data source.



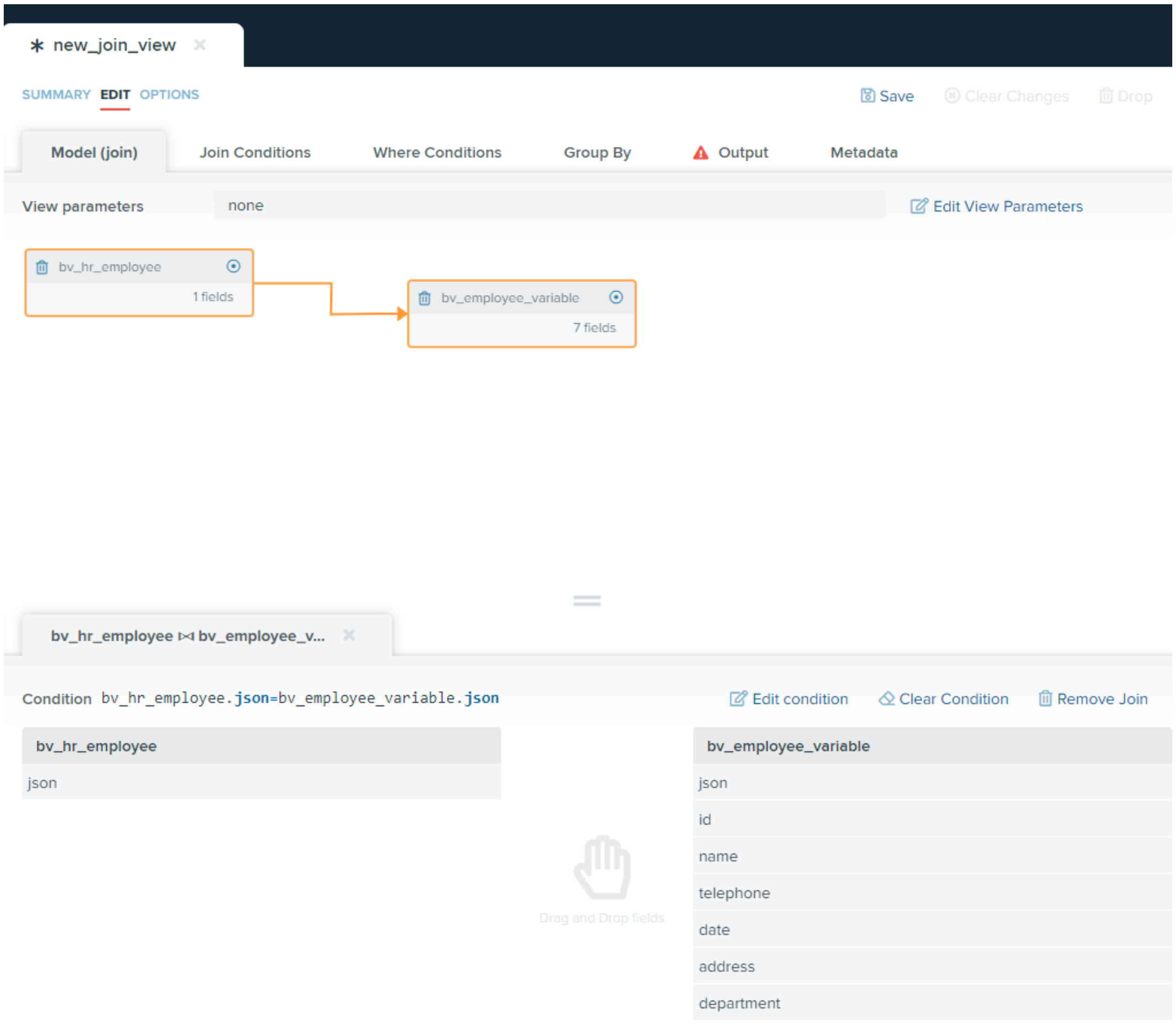
Database	admin	View name	bv_hr_employee
View type	Base	Cache status	Off
Data source	oracle		
Owner	admin	Last modifier	admin
Creation	5/25/2020 11:29 AM	Last modification	5/25/2020 11:29 AM
Folder	/		

View schema

PK	Field name	Field Type	Description
	json	text	

`bv_hr_employee` base view

Having all this done, it just remains one more step. We will finally create a derived view called dv_employee joining both base views bv_hr_employee and bv_employee_variable using bv_hr_employee.json = bv_employee_variable.json as a join condition. Thus, Virtual DataPort understands that we want to pass the json string of the JDBC base view as the input parameter of the From Variable one.



View parameters none

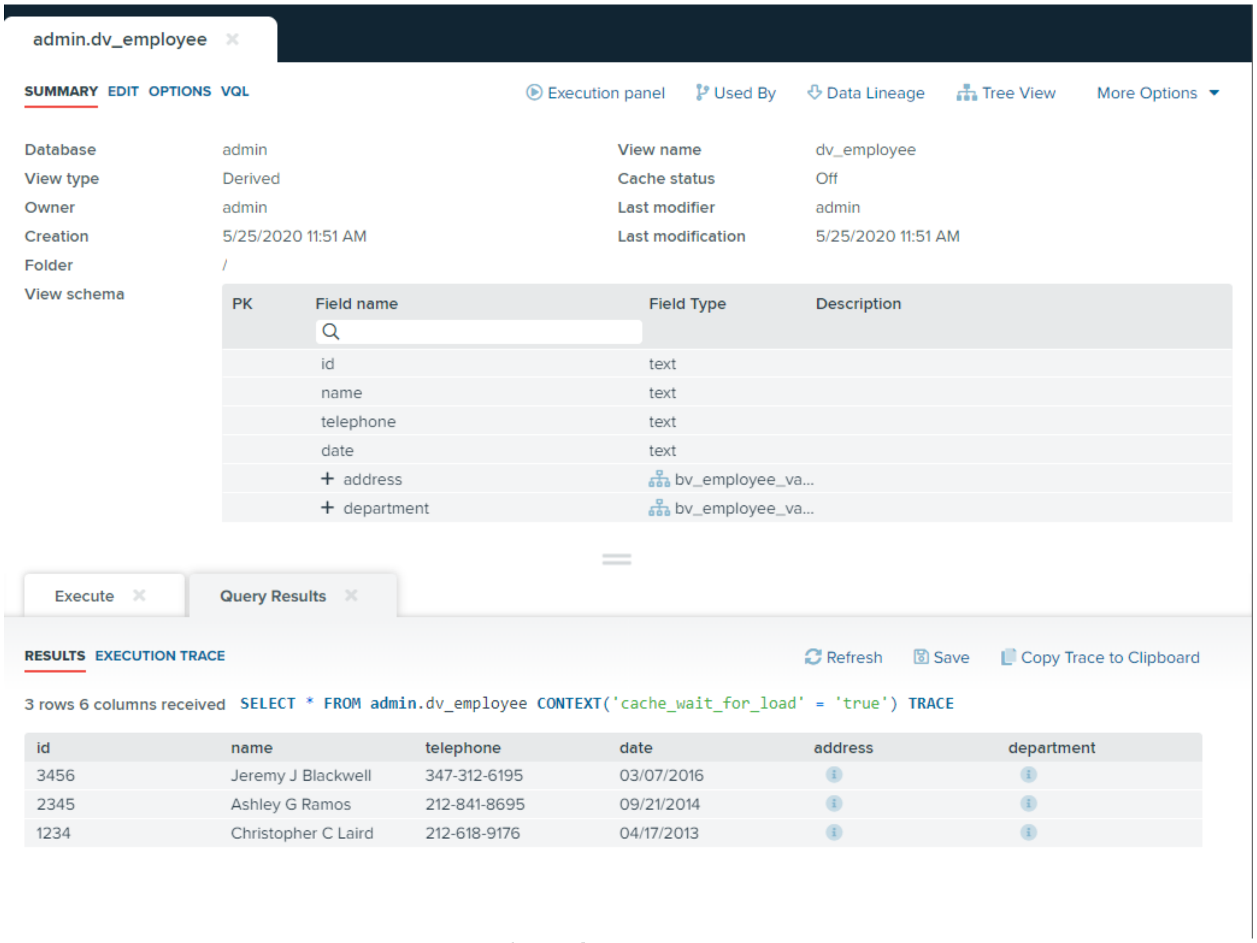
Condition bv_hr_employee.json=bv_employee_variable.json

bv_hr_employee	bv_employee_variable
json	json
	id
	name
	telephone
	date
	address
	department

Join details of dv_employee

In addition, the two json fields coming from both base views can be removed from the Output fields as they are not necessary any more, the whole json has been decomposed into individual fields.

The derived view dv_employee is ready to be executed and input parameters are no longer needed.


admin.dv_employee

SUMMARY EDIT OPTIONS VQL

Database	admin	View name	dv_employee
View type	Derived	Cache status	Off
Owner	admin	Last modifier	admin
Creation	5/25/2020 11:51 AM	Last modification	5/25/2020 11:51 AM
Folder	/		

View schema

PK	Field name	Field Type	Description
	id	text	
	name	text	
	telephone	text	
	date	text	
	+ address	bv_employee_va...	
	+ department	bv_employee_va...	

RESULTS EXECUTION TRACE

3 rows 6 columns received `SELECT * FROM admin.dv_employee CONTEXT('cache_wait_for_load' = 'true') TRACE`

id	name	telephone	date	address	department
3456	Jeremy J Blackwell	347-312-6195	03/07/2016	ⓘ	ⓘ
2345	Ashley G Ramos	212-841-8695	09/21/2014	ⓘ	ⓘ
1234	Christopher C Laird	212-618-9176	04/17/2013	ⓘ	ⓘ

Execution of `dv_employee`

In short, we have learned how helpful it can be to use a “From Variable" data source if we have decided to store JSON, CSV or XML in our information systems.

References

Virtual DataPort Administration Guide: Path from Variable
Denodo Knowledge Base: Accessing hierarchical JSON data in Hive from Denodo