How to detect changes in data sources

Revision 20231212

Denodo Inc.: 525 University Avenue, Suite 31, Palo Alto, CA 94301. USA www.denodo.com

CONTENTS

1 GOAL

This document describes how to detect changes in data sources combining the `GET_SOURCE_CHANGES` and `GET_VIEWS` stored procedures and how to automate the detection using Denodo Scheduler. This functionality can be used from the Denodo VDP Administration Tool clicking on the `Source Refresh` button of any base view.

2 CONTENT

2.1 GETTING THE MODIFIED FIELDS IN EVERY BASE VIEW

Important note: The instructions provided below have been tested in Denodo 7.0 and 8.0 versions. Please review the section: *"Compatibility with Denodo 6.0"* to get alternatives for the Denodo 6.0 version.

Denodo provides a stored procedure called GET_SOURCE_CHANGES that detects the differences between the current schema of a base view and its underlying data source. The syntax is:

```
GET_SOURCE_CHANGES (database_name : text, table_name : text)
```

This stored procedure returns a row for each field of the view and also a row for each field that is present in the data source, but not in the base view. If a field is an array or a register, there will also be a row for each one of its subfields.

Sometimes, if the virtual database has a lot of base views and the data source schemas are changing quite frequently, the manual execution of this stored procedure for every base view can be cumbersome.

It is possible to combine the GET_SOURCE_CHANGES and GET_VIEWS stored procedures to detect which fields of the views have changed.

You can create a new view database_change that will show the modified fields in all the base views of your VDP server. The following VQL can be executed to create that database_change view:

```
CREATE VIEW database_change AS
SELECT source_change.db_name AS db_name, source_change.table_name AS
table_name, source_change.field AS field, source_change.type AS type,
source_change.old_type AS old_type, source_change.modification AS
modification, source_change.depth AS depth
FROM GET_VIEWS() AS base_view
NESTED INNER JOIN GET_SOURCE_CHANGES() AS source_change ON
(base_view.input_view_type =0 and base_view.database_name =
source_change.db_name AND base_view.name = source_change.table_name)
WHERE source_change.modification <> '';
```

2.1.1 VQL Explanation:

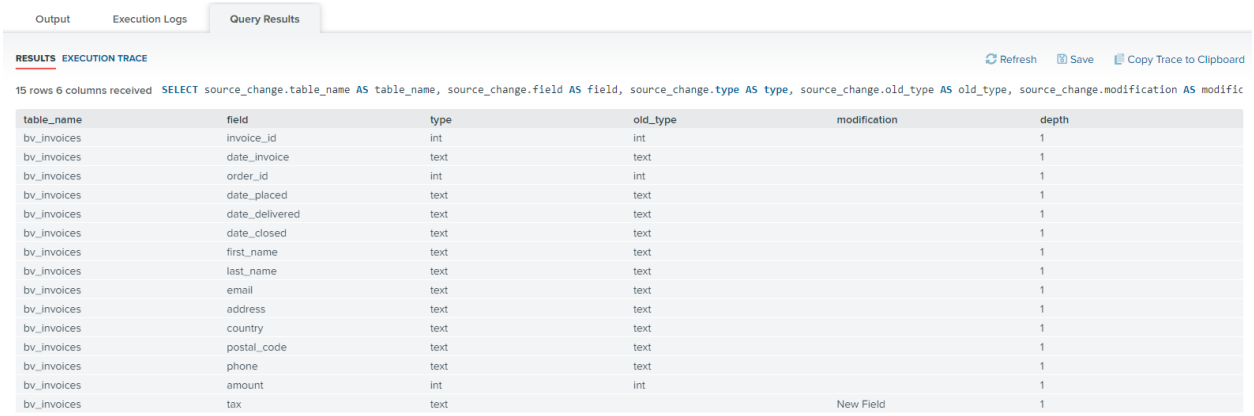
The next steps explain the operations done in the `database_change` view that has just been created executing the VQL:

1. The goal is to execute the procedure GET_SOURCE_CHANGES() for any base view and get the fields that have been modified from the output of that procedure.

2. A NESTED INNER JOIN is performed between the GET_VIEWS() and GET_SOURCE_CHANGES() stored procedures in order to ensure one execution of the GET_SOURCE_CHANGES per row in the GET_VIEWS() result. The join conditions are: `base_view.database_name = source_change.db_name AND base_view.name = source_change.table_name.`

3. In order to get just the base views from the GET_VIEWS() stored procedure, a `base_view.input_view_type =0` condition is added.

4. In addition to this, a filter is added to show only the modified fields. There is a field in the new view called "`modification`". This field describes the change detected in the field, for example:



Output | Execution Logs | Query Results

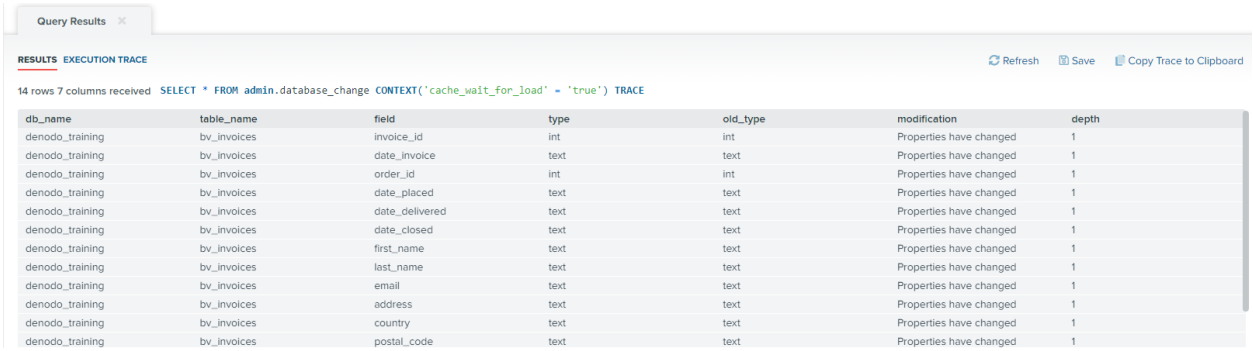
RESULTS | EXECUTION TRACE — Refresh | Save | Copy Trace to Clipboard

15 rows 6 columns received `SELECT source_change.table_name AS table_name, source_change.field AS field, source_change.type AS type, source_change.old_type AS old_type, source_change.modification AS modific`

table_name	field	type	old_type	modification	depth
bv_invoices	invoice_id	int	int		1
bv_invoices	date_invoice	text	text		1
bv_invoices	order_id	int	int		1
bv_invoices	date_placed	text	text		1
bv_invoices	date_delivered	text	text		1
bv_invoices	date_closed	text	text		1
bv_invoices	first_name	text	text		1
bv_invoices	last_name	text	text		1
bv_invoices	email	text	text		1
bv_invoices	address	text	text		1
bv_invoices	country	text	text		1
bv_invoices	postal_code	text	text		1
bv_invoices	phone	text	text		1
bv_invoices	amount	int	int		1
bv_invoices	tax	text		New Field	1

5. If "`modification`" is empty for a field it means that the field has not been modified. To add this filter, the following condition has been added to the VQL: `source_change.modification <> ''.`

6. Below an example of the output of the base view created with the VQL attached:



Query Results

RESULTS | EXECUTION TRACE — Refresh | Save | Copy Trace to Clipboard

14 rows 7 columns received `SELECT * FROM admin.database_change CONTEXT('cache_wait_for_load' = 'true') TRACE`

db_name	table_name	field	type	old_type	modification	depth
denodo_training	bv_invoices	invoice_id	int	int	Properties have changed	1
denodo_training	bv_invoices	date_invoice	text	text	Properties have changed	1
denodo_training	bv_invoices	order_id	int	int	Properties have changed	1
denodo_training	bv_invoices	date_placed	text	text	Properties have changed	1
denodo_training	bv_invoices	date_delivered	text	text	Properties have changed	1
denodo_training	bv_invoices	date_closed	text	text	Properties have changed	1
denodo_training	bv_invoices	first_name	text	text	Properties have changed	1
denodo_training	bv_invoices	last_name	text	text	Properties have changed	1
denodo_training	bv_invoices	email	text	text	Properties have changed	1
denodo_training	bv_invoices	address	text	text	Properties have changed	1
denodo_training	bv_invoices	country	text	text	Properties have changed	1
denodo_training	bv_invoices	postal_code	text	text	Properties have changed	1

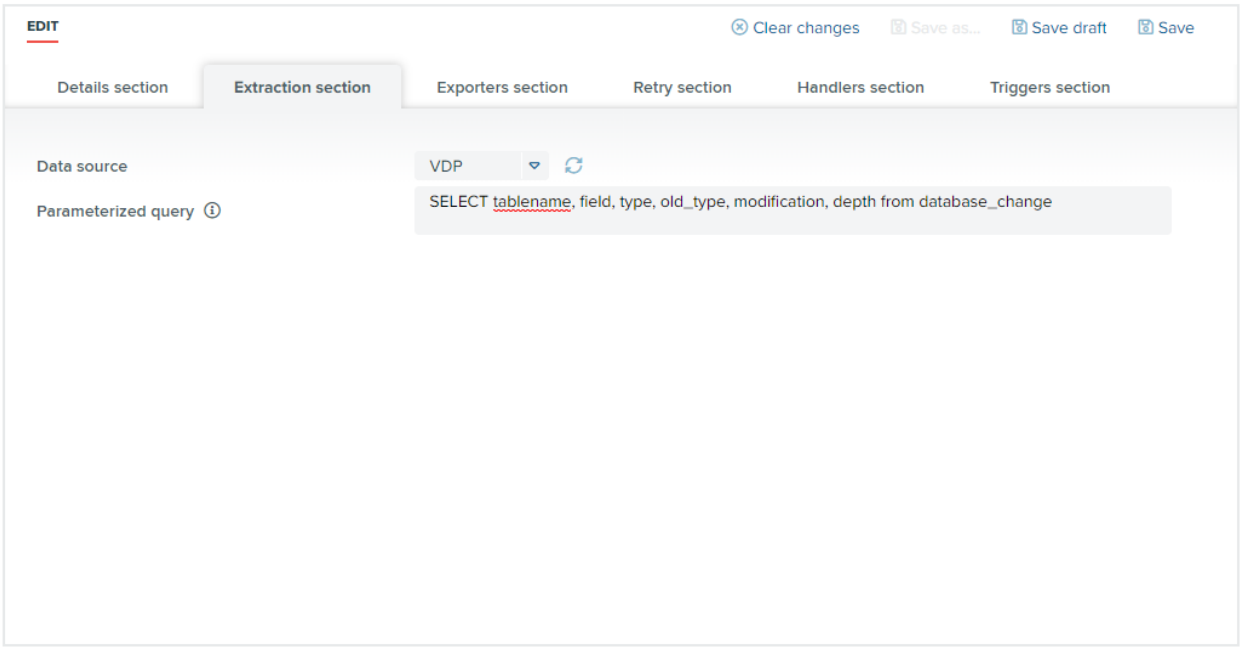
2.2 AUTOMATING THE DETECTION OF THE CHANGES IN SCHEDULER

Once we have a view that returns the list of changes in the data sources of Virtual DataPort database, we can create a Scheduler job to automate the detection of these changes. Go to the Scheduler Administration Tool and follow these steps:

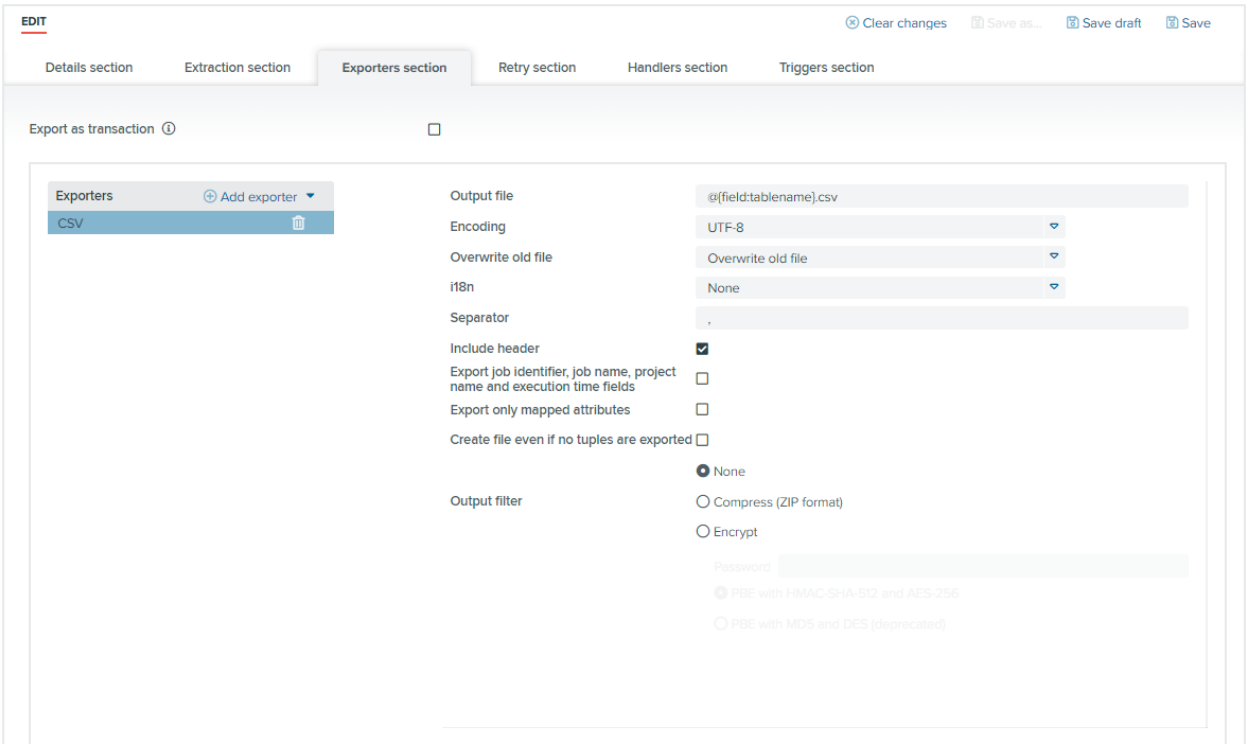
1. Create a new job of type "VDP".

2. Fill the Extraction section as follows:
   a. In the data source field, select a data source that points to the Virtual DataPort database where the 'database_change' view has been created.
   b. In the "Parameterized query" field, use the following query

```
SELECT table_name, field, type, old_type, modification, depth
FROM database_change
```



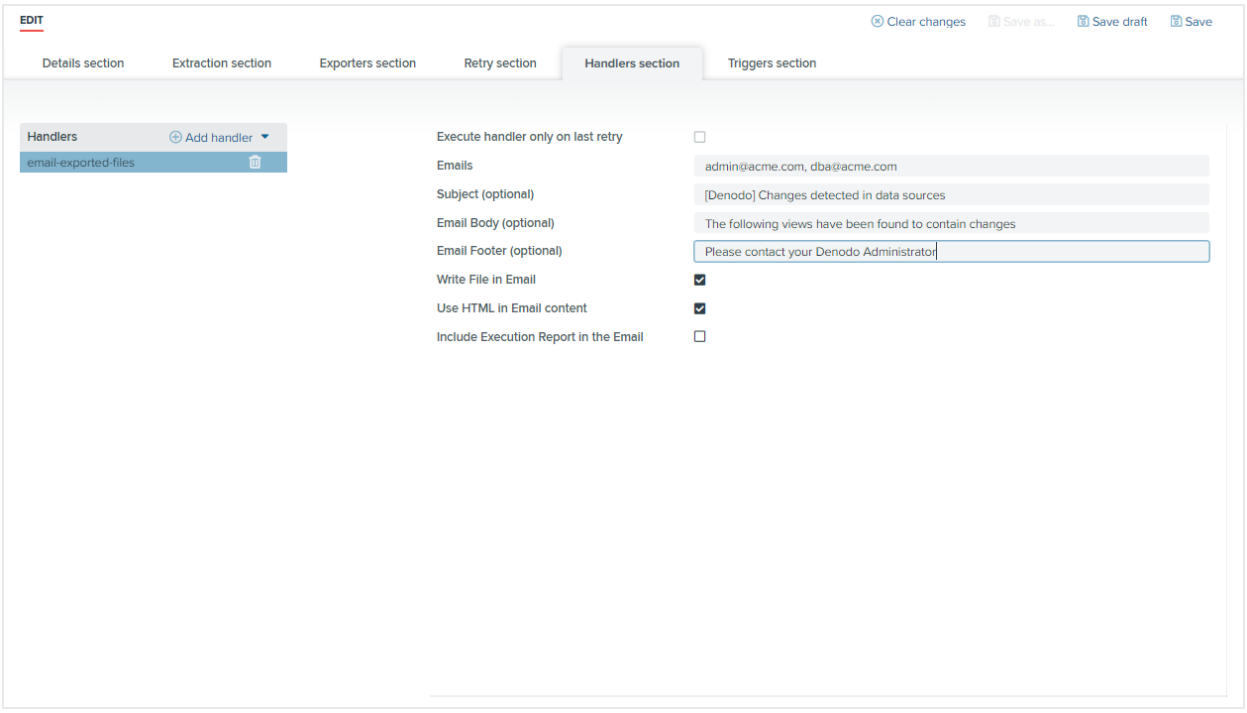
3. Go to the Exporters section and add a CSV exporter. Configure it as follows:
   a. Use the following as Output file name: @{field:tablename}.csv.
   b. Select UTF-8 as encoding.
   c. Select "Overwrite old file".
   d. Use comma (,) as separator.
   e. Select the "Include header" option.
   f. Leave all the other parameters with their default values.



4. To receive email notifications that include the CSV file generated in the previous step a Denodo Connects tool called "**Denodo Custom Handle Email Exported Files**" can be used. This plug-in can be found in the Denodo Support Site under the Resources > Denodo Connects section. Follow the instructions in the Custom Handler Email Exported Files - User Manual to install it.

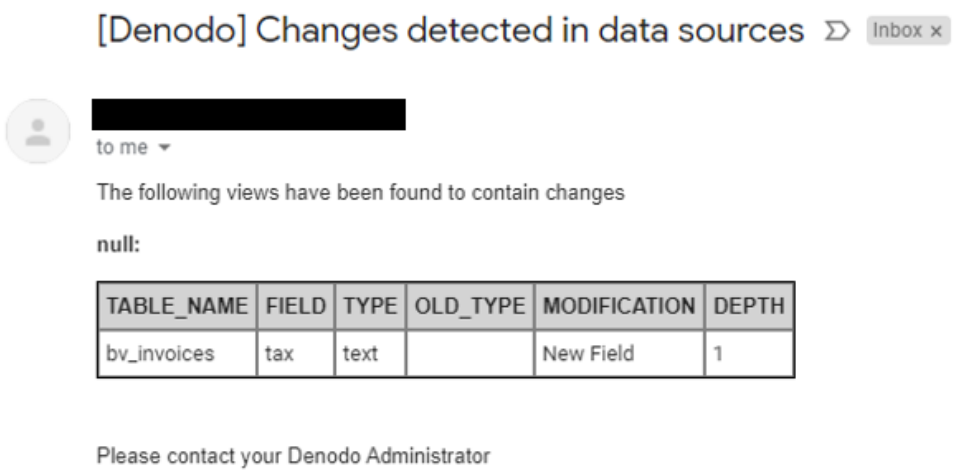
   In order to send emails, Denodo Scheduler needs to be also properly configured with the values of an email server. You can find more details in the Mail Settings section of the Scheduler Administration Guide.

5. Once the plugin is installed go to the `Handlers section` of the job and add a new "`email-exported-files`" handler. Configure it as follows:
   a. In the `Emails` test box, add the values of the emails that will receive the notification separated by commas.
   b. Add Values for the `Subject`, `Body` and `Footer` according to the notification emails.
   c. Check `Write File in Email.`
   d. Check `Use HTML in Email content.`
   e. Leave the other options as default .



6. In the `Triggers section` specify the time expression that sets when the analysis will be executed. It uses a UNIX "cron expression". You can find more details on how to configure this screen in the Time-based Job Scheduling section of the Scheduler Administration Guide.

7. Save the job. It can be executed manually by clicking on the "`Play`" icon next to it. If there is any change an email outlining the changes similar to this will be sent:



2.3 COMPATIBILITY WITH DENODO 6.0

The Denodo 6.0 version does not include the GET_SOURCE_CHANGES and GET_VIEWS stored procedures. As an alternative, you can use SOURCE_CHANGES and CATALOG_VIEWS stored procedures. Keep in mind that the syntax for using these procedures may be different from the one explained in the document.

In addition, the procedures available in Denodo 6.0 only return results from the views of the database where the query is executed. If you want to automatically detect changes for different databases, you need to create different data sources in Scheduler pointing to every database and then create a new job for each data source.

3 REFERENCES

Virtual DataPort VQL Guide: GET_SOURCE_CHANGES
Virtual DataPort VQL Guide: GET_VIEWS
Scheduler Administration Guide: Server Configuration - Mail configuration
Scheduler Administration Guide: Time-based Job Scheduling Section
Denodo Connects: Denodo Distributed File System Custom Wrapper - User Manual
Virtual DataPort VQL Guide: CATALOG_VIEWS (Denodo 6.0)
Virtual DataPort VQL Guide: SOURCE_CHANGES (Denodo 6.0)