How To Automate Privileges Assignment Based On Custom Criteria

Revision 20210824

NOTE
This document is confidential and proprietary of **Denodo Technologies**.
No part of this document may be reproduced in any form by any means without prior written authorization of **Denodo Technologies**.

Copyright © 2023
Denodo Technologies Proprietary and Confidential

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

CONTENTS

1 INTRODUCTION

This article gives some examples on how to assign privileges to roles or users automatically, based on some property of the views to be accessed.

The Denodo privileges system acts at the database or element level. This means that there is no concept of folder privileges. However, through the usage of the predefined stored procedures that query the Virtual DataPort metadata combined with Scheduler, we can automate the privileges assignment even at the folder level.

The most typical example that surfaces often is about applying the same set of privileges on all the views placed in the same folder (Case 1). In the other scenarios treated in this article we will see how to apply the same set of privileges to all interface views, the ones that are typically exposed to the data consumers (Case 2). Then we will explain how to apply a masking rule on all the views that contain a given field (Case 3). Finally, we will see how to apply a row restriction when the view description contains a given tag (Case 4).

Throughout the read of this article, and while generalizing these examples to your needs, you may find it useful to keep an eye on the privileges granting statements syntax as a reference.

2 CASE 1: APPLY PRIVILEGES TO ALL THE VIEWS IN A FOLDER

Let's say that you want to give read access to all users assigned to role `finalusers` to all the views in the folder `/published` in database `mydb`. This is a typical scenario in which all the views in this folder must be public for final consumers.

Keep in mind the following considerations:

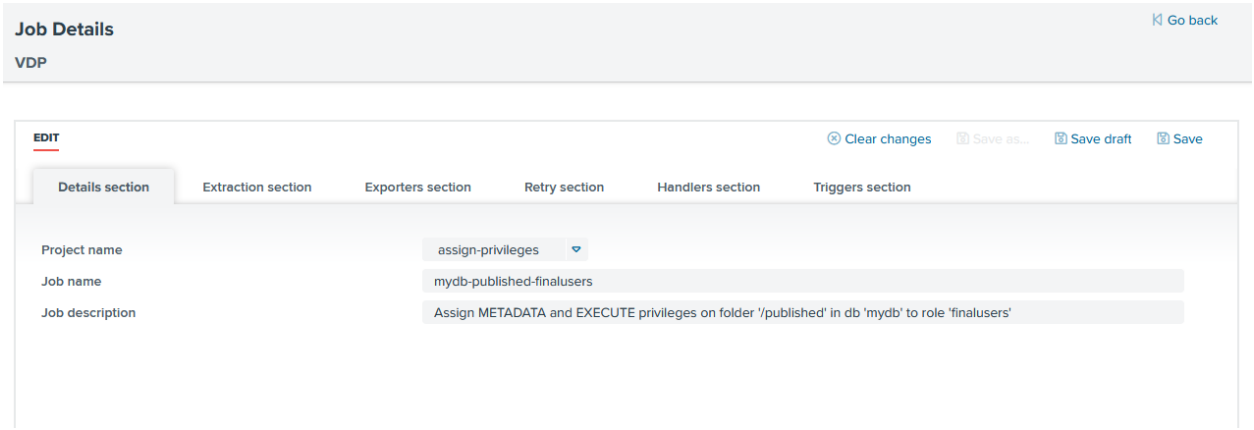
- To give a role read access to a view, you must give the `EXECUTE` and `METADATA` privileges over it. As granting `EXECUTE` implicitly grants `METADATA`, we can just specify `EXECUTE` in the VQL statement.
- The role you want to assign the privileges on, must have the `CONNECT` privilege on the database in which the folder lives.
- The syntax of the VQL statement to assign privileges is:

```
ALTER ROLE <role name> GRANT <Privilege1,Privilege2,...,PrivilegeN> ON <view name>
```

Then, the high-level procedure to automatically assign privileges would consist in defining a Scheduler job that executes, with an appropriate frequency, a series of dynamically generated `ALTER ROLE` statements obtained by querying the metadata catalog, via the `get_views()` and `catalog_permissions()` stored procedures.

This is the step-by-step procedure in the case where the role is called `finalusers`, the database is called `mydb` and the folder is `/published`.

1. **Create a Scheduler Job of type VDP**. Assign it to an appropriate project and give it a name and description. Remember that the project should have at least one data source of type VDP.



2. In the *Extraction section*, specify a variable called `@ALTER_ROLE_VQL`.

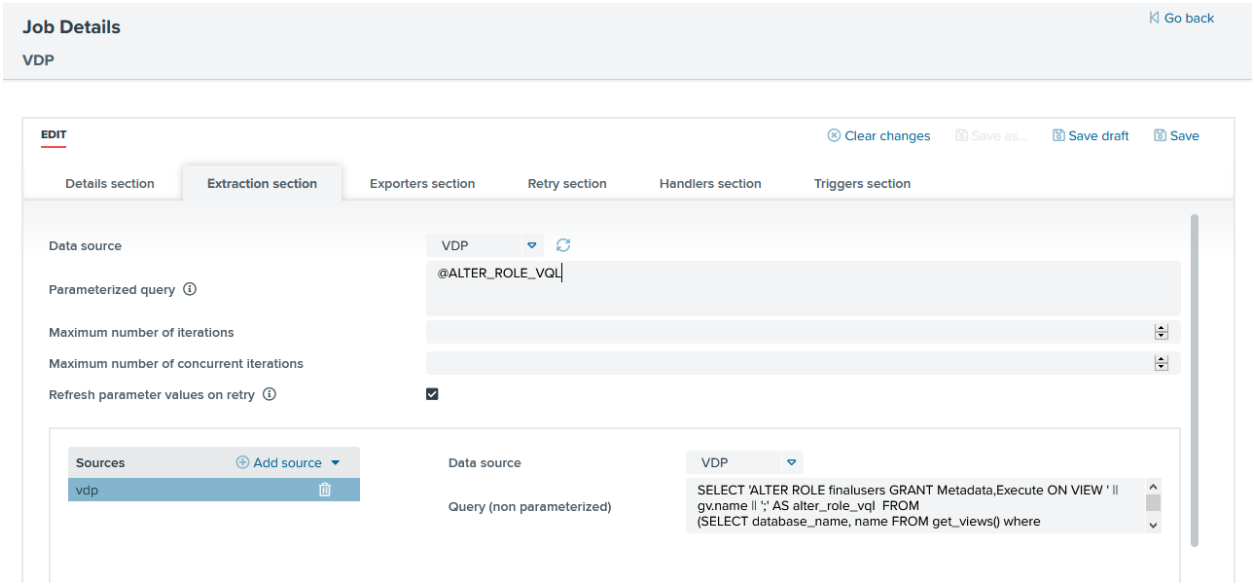
3. Click on Add source then select type `vdp` and choose your data source. In the *`Query (non parameterized)`* field, specify the query:

```
SELECT 'ALTER ROLE finalusers GRANT Execute ON ' || database_name || '.' ||
name                ||';'                AS                alter_role_vql
                                    FROM     get_views()     as     gv
                                WHERE     database_name=     'mydb'
                                        AND     folder='/published'
        AND type='view';
```

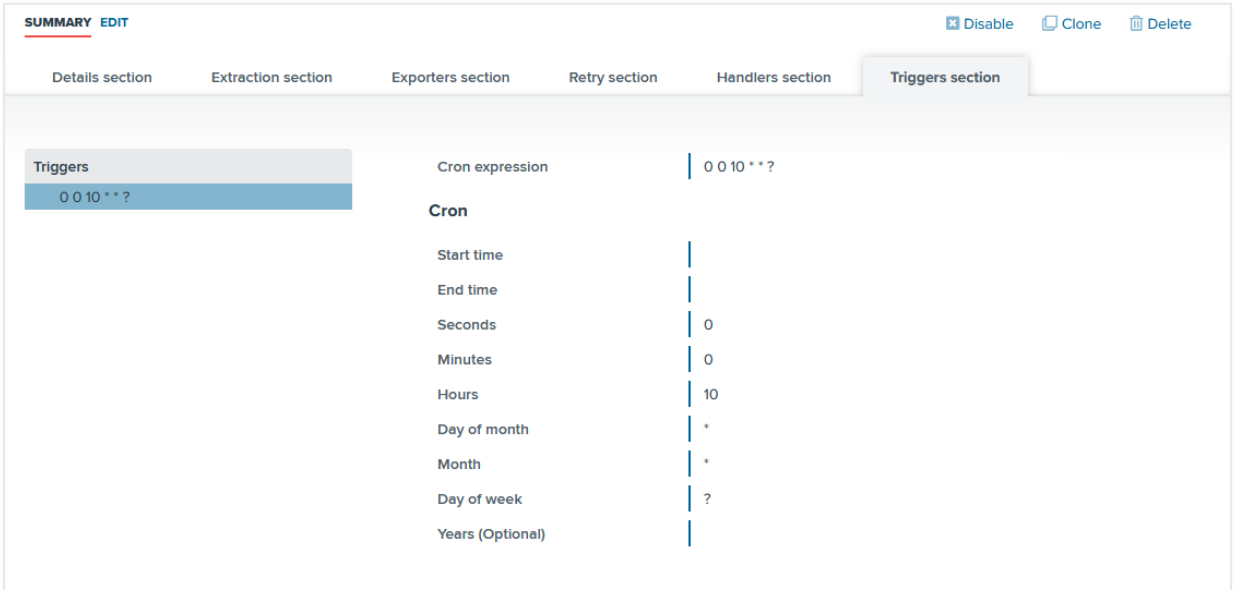
With this query you are forming a valid `ALTER ROLE VQL` statement per view in folder `/published`. Each of these statements will be executed when the scheduler job is run, meaning that the statement does not have any effect on those views with the privileges already assigned. If you have a very large number of views in the folder you may want to avoid running `ALTER ROLE` statements on those views. This can be achieved by joining the views resulting from the stored procedures `GET_VIEWS` and `CATALOG_PERMISSIONS`:

```
SELECT 'ALTER ROLE finalusers GRANT EXECUTE ON ' || database_name || '.' ||
gv.name                ||';'                AS                alter_role_vql
FROM
(SELECT                database_name,                name
                                        FROM      get_views()
                                    WHERE      database_name='mydb'
                                        AND     folder='/published'
                                            AND     type='view')     gv
LEFT                                OUTER                                JOIN
                (SELECT    DISTINCT    userrolename,    dbname,    elementname
                                        FROM     catalog_permissions()
```

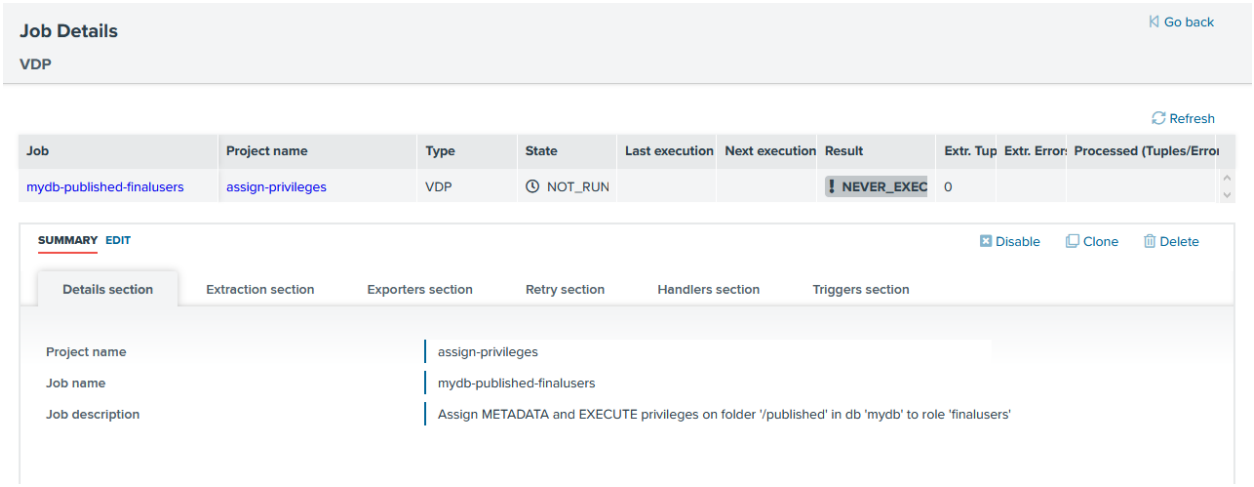
```
                                        WHERE     userrolename='finalusers'
                                          AND     dbname='mydb'
                                  AND     elementname     IS     NOT     NULL
                                          AND     elementtype='View')     cp
ON     gv.database_name     =     cp.dbname     AND     gv.name=cp.elementname
WHERE cp.elementname IS NULL;
```



4. **Schedule the job** according to your needs. The scheduling must be configured via a cron expression in the `Triggers section`.



5. **Save the job**.



6. Test the job by performing the following tasks:
    a. Create a new view in the folder `/published`.
    b. Connect to Virtual DataPort with a user with role `finalusers` and check that you don't see the view just created.
    c. Run the Scheduler job just created.
    d. Refresh or connect to the Virtual DataPort with a user with role `finalusers`. Now you should see the view just created.
    e. If you get an error or an unexpected result at step c, the best way to debug is to copy the query above in a VQL Shell and edit / run until you get the correct `ALTER ROLE` statement.

3 CASE 2: APPLY PRIVILEGES ON ALL THE INTERFACE VIEWS

The query specified at step 3 in Case 1 can be modified according to your needs: we can change the privileges to be applied (`METADATA`, `EXECUTE` …) as well as the objects which we want to apply the privileges on.

In this case, we would like to again apply the `METADATA` and `EXECUTE` privileges on all the views of type interface.

```
SELECT 'ALTER ROLE finalusers GRANT EXECUTE ON ' || database_name || '.' ||
gv.name                                                              ||';'
                                        FROM     get_views()     as      gv
                                       WHERE     database_name='mydb'
                                           AND     folder='/published'
                                                       AND      type='view'
         AND view_type = 2
```

The only difference with the query at step 4 in Case 1 is an additional filtering condition that outputs just the interface views (`view_type=2`).

4 CASE 3: MASK A FIELD IN ALL THE VIEWS THAT CONTAIN A SPECIFIC FIELD NAME

Let us now study the scenario where we want to apply the masking of the column `ssn` on all the derived views in database `mydb` that contain that field for users assigned to role `finalusers`.

For each view that satisfies those conditions we need to run two **`ALTER ROLE`** statements; the first one is the same as in the cases above, while the second one applies the masking rule.

The syntax of the masking rule statement is as follows:

```
ALTER ROLE <role name> GRANT EXECUTE ANY(<field to be masked>) THEN <masking condition> MASKING ON <view to be masked>;
```

In our specific scenario, the whole query would be the following:

```
SELECT 'ALTER ROLE finalusers GRANT EXECUTE ON ' || database_name || '.' || name || ' GRANT EXECUTE WHEN ANY(ssn) THEN ' || '''1<>1''' || ' MASKING ON ' || database_name || '.' || name ||';' AS alter_role_vql FROM GET_VIEWS() WHERE name in (SELECT distinct view_name FROM GET_VIEW_COLUMNS() WHERE column_name = 'ssn' AND input_database_name = 'mydb') AND input_database_name = 'mydb' AND view_type = 1;
```

Once again, this query can be automatized with the technique described in Case 1.

5 CASE 4: RESTRICT ROWS VISIBILITY FOR VIEWS WITH A SPECIFIC TAG OR KEYWORD IN THEIR DESCRIPTION

In this case, we are in the scenario where we would like to restrict the records that users assigned to a given role can see on all the views that include a given text in the view description.

Specifically, in this example we would like to show only the rows with `cc_company=1` to users assigned to role `salesanalysts_cc1` for all views that contain the tag `#DataCorp` in their description. This assumes a scenario where DataCorp code is 1 (`cc_company=1`) and the development convention dictates to add this tag to all views that should be executable by analysts belonging to that organization.

Again, we just need to change the query in step 3 described in Case 1 with the following:

```
SELECT
'ALTER ROLE salesanalyst_cc1 GRANT EXECUTE ON ' || database_name || '.' ||
name || ' GRANT EXECUTE WHEN() THEN ''cc_company = 1'' ON ' || database_name
||       '.'       ||       name       ||       ';'       AS       alter_role_vql
                                                    FROM             GET_VIEWS()
     WHERE input_database_name = 'sandbox'
       AND description LIKE '%#DataCorp%';
```

6 CONCLUSION

In this article we have seen a privileges assignment technique that leverages metadata-querying predefined stored procedures (**GET_VIEWS**, **GET_VIEW_COLUMNS**) along with parameterized queries in Denodo Scheduler to apply your privileges policy in an automated manner. The idea is to dynamically construct the proper **ALTER ROLE** statement on the views that match some user-defined criteria. This pattern can be customized to match your needs.