



Denodo Incremental Cache Load Stored Procedure - User Manual

Revision 20180619

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 © 2018
Denodo Technologies Proprietary and Confidential

CONTENTS

1 OVERVIEW.....	3
2 IMPORTING THE STORED PROCEDURE INTO VIRTUAL DATAPORT.....	4
2.1 IMPORTING THE EXTENSION TO THE VIRTUAL DATAPORT SERVER.....	4
2.2 ADDING THE STORED PROCEDURE USING VQL LANGUAGE.....	4
2.3 ADDING THE STORED PROCEDURE USING THE VIRTUAL DATAPORT ADMINISTRATION TOOL MENU.....	4
3 GENERAL PROCESS AND REQUIREMENTS.....	6
4 INVOKING THE DENODO INCREMENTAL CACHE LOAD STORED PROCEDURE.....	7
4.1 EXAMPLE OF EXECUTION AND SCHEDULING.....	9
4.2 USE OF @LASTREFRESHCACHE VARIABLE.....	9
5 LIMITATIONS.....	10

1 OVERVIEW

Virtual DataPort incorporates a system called cache module that can store a local copy of the data retrieved from the data sources, in a JDBC database. This may reduce the impact of repeated queries hitting the data source and speed up data retrieval, especially with certain type of sources.

The Cache Engine, available in Virtual DataPort, allows two main modes: Partial and Full. When you enable the first one, the cache only stores some of the tuples of the view and, at runtime, when a user queries a view with this cache mode, the server checks if the cache contains the data required to answer the query. If it does not have this data, the server queries the data source. However, if you use the second one, the data of the view is always retrieved from the cache database instead of from the source.

When you use the Full cache option, you can make it incremental, specifying a condition to retrieve the data modified since the last cache refresh.

The Denodo Incremental Cache Load Stored Procedure is focused on giving support to this last type of cache updating, making this process executable on demand with a stored procedure that will update the cache based on the condition the user gives.

2 IMPORTING THE STORED PROCEDURE INTO VIRTUAL DATAPORT

2.1 IMPORTING THE EXTENSION TO THE VIRTUAL DATAPORT SERVER

For running the Denodo Incremental Cache Load Stored Procedure you have to load the `denodo-incremental-cache-load-{vdpversion}-{version}-jar-with-dependencies.jar` file first of all, using the Jar Management option of VDP Administration Tool located in the File > Jar Management menu (or File > Extension management in Denodo 7.0). Once you have imported the Jar file you can add the stored procedure.

2.2 ADDING THE STORED PROCEDURE USING VQL LANGUAGE

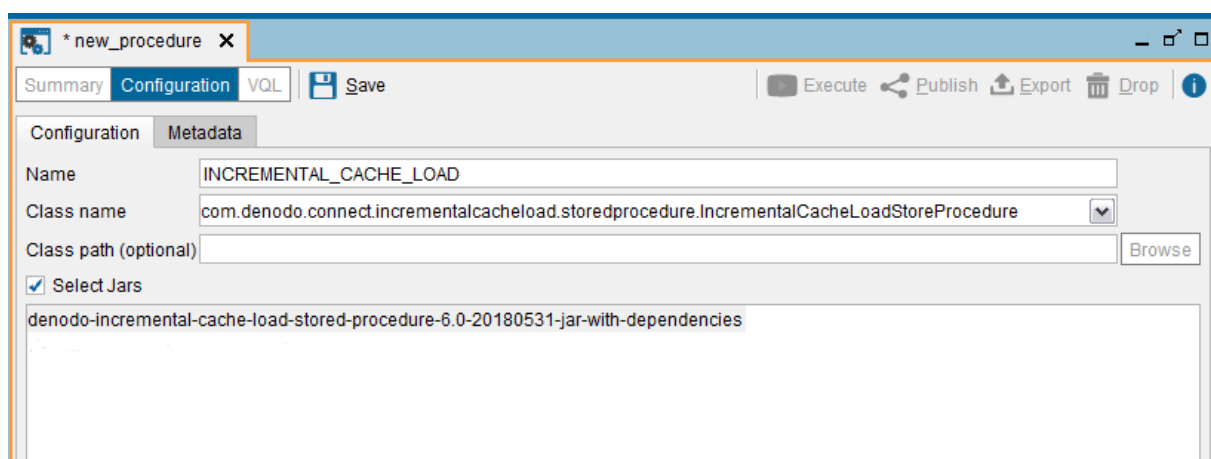
You can add a new stored procedure with the statement CREATE PROCEDURE:

```
CREATE [OR REPLACE] PROCEDURE <name:identifier>
CLASSNAME='com.denodo.connect.incrementalcacheload.storedprocedure.Incremental
CacheLoadStoreProcedure'
    JARS 'denodo-incremental-cache-load-<vdpversion>-<version>';
[ FOLDER = <literal> ]
[ DESCRIPTION = <literal> ]
```

The `classname` must be `com.denodo.connect.cache.incrementalcacheload.storedprocedure.IncrementalCacheLoadStoreProcedure` and the JARS value must be the Jar file, `denodo-incremental-cache-load-<vdpversion>-<version>`, previously added to the Virtual DataPort Server (see the [Importing the extension to the Virtual DataPort Server section](#) for more information).

2.3 ADDING THE STORED PROCEDURE USING THE VIRTUAL DATAPORT ADMINISTRATION TOOL MENU

You can add a new stored procedure clicking Stored procedure on the menu File > New:

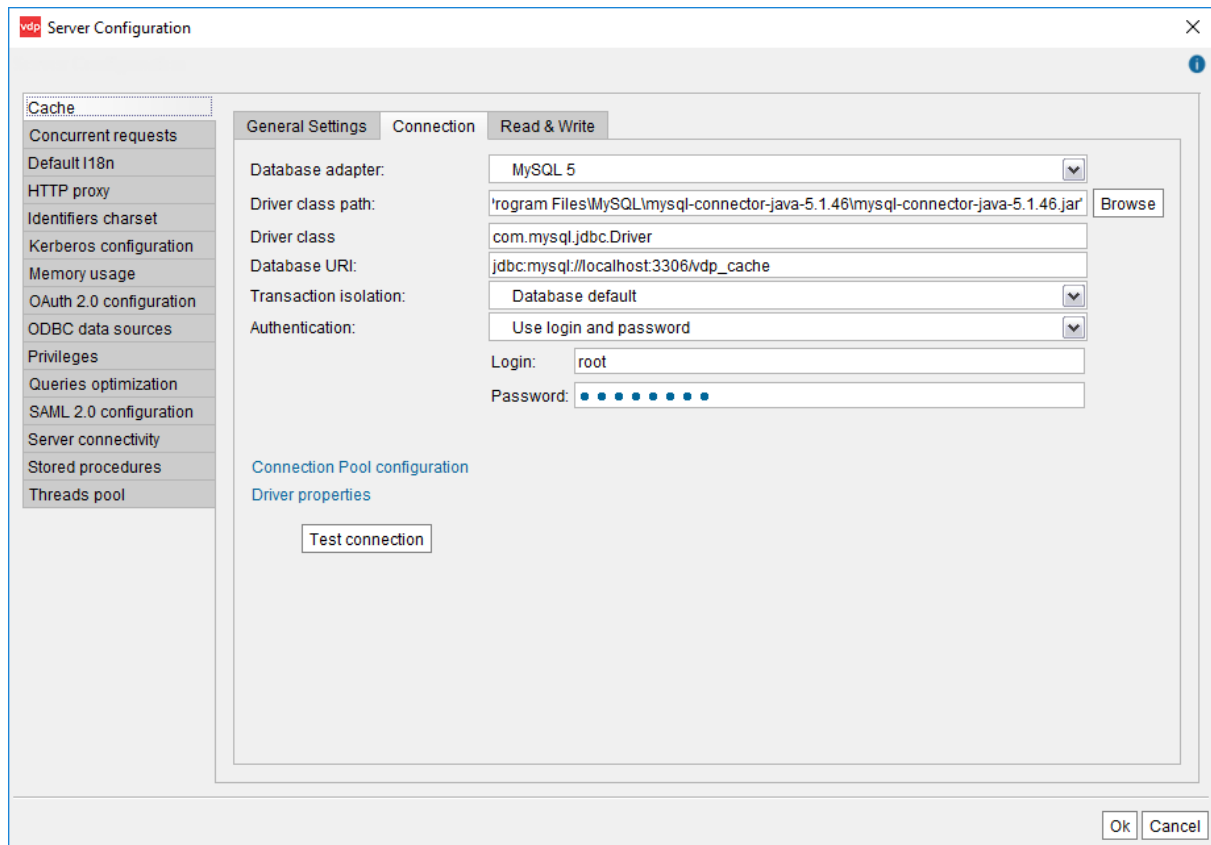


The screenshot shows the Denodo Administration Tool interface with the 'new_procedure' window open. The 'Configuration' tab is selected, and the 'Metadata' sub-tab is active. The 'Name' field is set to 'INCREMENTAL_CACHE_LOAD'. The 'Class name' dropdown is set to 'com.denodo.connect.incrementalcacheload.storedprocedure.IncrementalCacheLoadStoreProcedure'. The 'Class path (optional)' field is empty, with a 'Browse' button next to it. The 'Select Jars' checkbox is checked, and a list of jars is displayed, including 'denodo-incremental-cache-load-stored-procedure-6.0-20180531-jar-with-dependencies'.

You must set a name in the Name field and select the Select Jars checkbox in order to use the Jar file, denodo-incremental-cache-load-<vdpversion>-<version>, previously added to the Virtual DataPort Server (see the **Importing the extension to the Virtual DataPort Server section** for more information).

3 GENERAL PROCESS AND REQUIREMENTS

When you enable the Cache Engine in the Virtual DataPort the default DBMS (Database Management System) is the embedded Apache Derby database but it is highly recommended to change it and use an external one, specially when you are working with high amounts of data. You can use many different DBMS and you can configure it in Administration > Server Configuration > Cache > Connection. Here is an example of a cache database configured over MySQL 5:



The screenshot shows the 'Server Configuration' window with the 'Cache' section selected in the left sidebar. The 'Connection' tab is active, displaying the following configuration:

- Database adapter:** MySQL 5
- Driver class path:** 'rogram Files\MySQL\mysql-connector-java-5.1.46\mysql-connector-java-5.1.46.jar' (with a 'Browse' button)
- Driver class:** com.mysql.jdbc.Driver
- Database URI:** jdbc:mysql://localhost:3306/vdp_cache
- Transaction isolation:** Database default
- Authentication:** Use login and password
- Login:** root
- Password:** (masked with dots)

Below the configuration fields, there are links for 'Connection Pool configuration' and 'Driver properties', and a 'Test connection' button. The 'Ok' and 'Cancel' buttons are at the bottom right.

4 INVOKING THE DENODO INCREMENTAL CACHE LOAD STORED PROCEDURE

The stored procedure needs several input parameters:

- **DATABASE_NAME** (non-nullable text): the Virtual DataPort's database name where you have the view with the data that you are going to cache.
- **VIEW_NAME** (non-nullable text): the view name that the Denodo Incremental Cache Load Stored Procedure will cache.
- **LAST_UPDATE_CONDITION** (non-nullable text): the condition that filters the tuples to be updated. It will be used in the WHERE clause of the query to retrieve the data from the source to the cache system. A simple example could be: 'registered_date > ''2014-12-31''' (note that you have to escape the single quotes).
- **NUM_ELEMENTS_IN_CLAUSE** (non-nullable text): a number that indicates the chunk size of the IN clauses in the queries that are going to update the cache. This parameter is used to try to get the best performance, as this procedure may have to move a very large amount of data and the time elapsed will depend directly on this variable. It has to be greater than 0 and is limited depending on every database system own limitations. As every scenario is different, it's hard to propose an optimal chunk size. However, chunks between 5.000 and 20.000 get the best performances.

After installing the stored procedure there are four ways of invoking it:

1. Using the Execute button in the dialog that displays the schema of the stored procedure. The tool will show a dialog where you have to enter the input values.

Stored procedure parameters

database_name	text		<input type="checkbox"/> Set as null
view_name	text		<input type="checkbox"/> Set as null
last_update_condition	text		<input type="checkbox"/> Set as null
num_elements_in_clause	text		<input type="checkbox"/> Set as null

☒ Limit rows
☐ Stop query when the limit is reached
 ☐ Open results in new tab

Ok

Cancel

2. With the CALL statement :

```
CALL INCREMENTAL_CACHE_LOAD('database_name', 'view_name', 'id < 20', 10000);
```

3. Invoking the procedure on the FROM clause of a SELECT statement:

```
SELECT *
FROM INCREMENTAL_CACHE_LOAD('database_name', 'view_name', 'id < 20', 10000);
```

4. Invoking the procedure on the FROM clause of a SELECT statement but providing the input parameters on the WHERE clause:

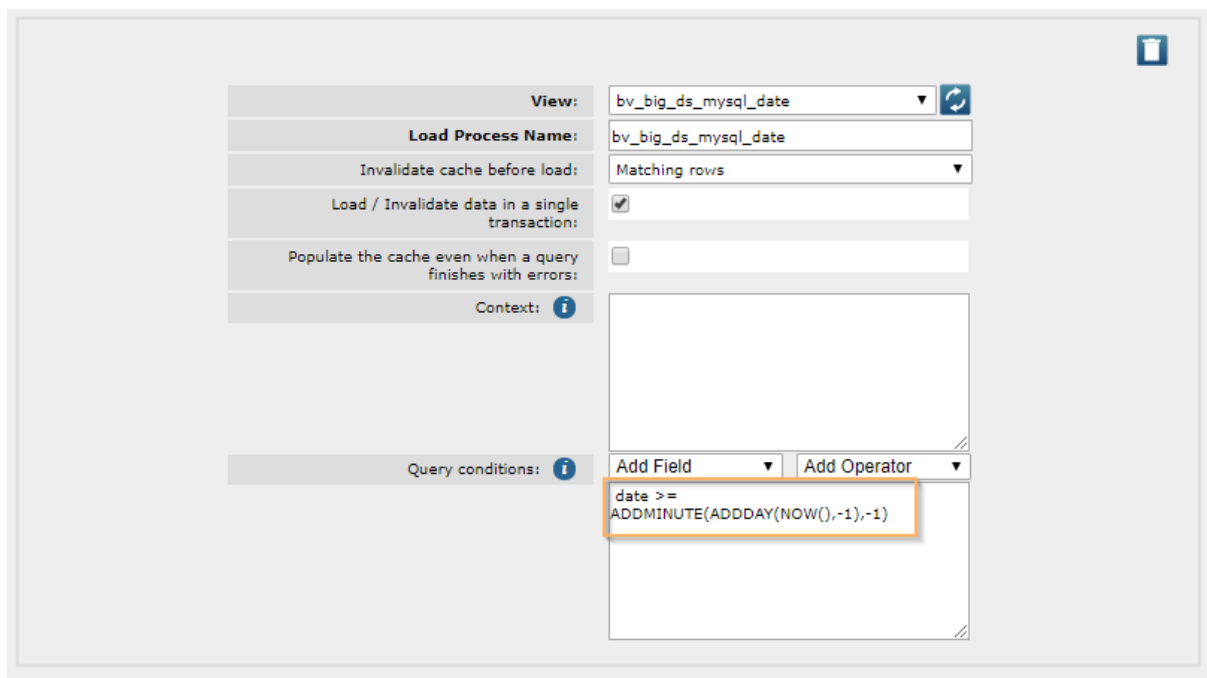
```
SELECT *
FROM INCREMENTAL_CACHE_LOAD()
WHERE DATABASE_NAME = 'database_name'
      and VIEW_NAME = 'view_name'
      and LAST_UPDATE_CONDITION = 'id < 20'
      and NUM_ELEMENTS_IN_CLAUSE = 10000;
```

The Denodo Incremental Cache Load Stored Procedure also has an output parameter called NUM_UPDATED_ROWS that indicates the number of rows updated in the cache system.

4.1 EXAMPLE OF EXECUTION AND SCHEDULING

As the use of the interpolation variable @LASTCACHEREFFRESH is not currently supported in this procedure there is the possibility of, for instance, defining a new daily job in the Scheduler by using the condition '**lastmodifieddate >= ADDMINUTE(ADDDAY(NOW(), -1), -1)**' in the LAST_UPDATE_CONDITION parameter. Of course, you can adjust the formula to make the process run hourly, weekly..

To add this new job, you need to log in Scheduler, access Job > New Job > VDPCache and define the condition in the Extraction section as follows:



4.2 USE OF @LASTREFRESHCACHE VARIABLE

The Denodo Incremental Cache Load Stored Procedure supports the use of the @LASTCACHEREFFRESH interpolation variable, which allows to create update cache conditions based on its last data load. An example of execution using this variable could be the following:

```
call incremental_cache_load('test', 'bv_big_ds_mysql', 'date >= @LASTCACHEREFFRESH', '1000');
```

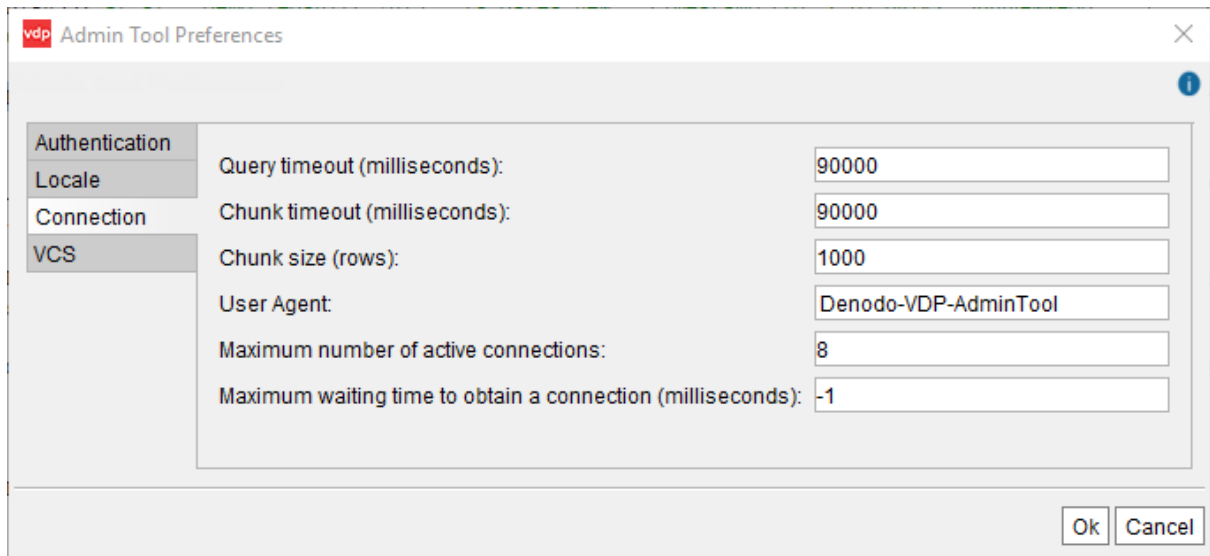
5 LIMITATIONS

Synchronous process

The execution of Denodo Incremental Cache Load Stored Procedure is synchronous. The stored procedure prevents Virtual DataPort Server from processing the data but it will wait until the end of the process.

Timeout considerations

The process of loading the cache can be quite long, depending on how much data you need to move. If you invoke the Denodo Incremental Cache Load Stored Procedure using the Administration Tool you must revise the Admin Tool Preferences where you can change the query timeout. If the value 0 is specified, the tool will wait indefinitely until the process ends.



Privileges over GET_PRIMARY_KEYS() stored procedure

The Denodo Incremental Cache Load Stored Procedure uses internally another stored procedure, GET_PRIMARY_KEYS(), that has some limitations depending on the privileges granted to the user that runs it. If the user is not an administrator user, consider the following:

- If the parameter input_database_name (DATABASE_NAME on our stored procedure) is not NULL, the procedure returns an error if the user does not have CONNECT privileges over this database.
- The procedure will only not return information about the primary keys of views over which the user has READ privileges.