



Denodo Incremental Cache Load Stored Procedure - User Manual

Revision 20190204

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 © 2019
Denodo Technologies Proprietary and Confidential

CONTENTS

1 OVERVIEW.....	3
2 INSTALLATION.....	4
2.1 IMPORTING THE STORED PROCEDURE.....	4
2.2 ADDING THE STORED PROCEDURE: VQL SHELL.....	4
2.3 ADDING THE STORED PROCEDURE: VIRTUAL DATAPORT ADMINISTRATION TOOL MENU.....	4
3 REQUIREMENTS.....	5
4 EXECUTION.....	7
4.1 USE OF DENODO SCHEDULER.....	9
4.2 USING THE SP WITH DERIVED VIEWS.....	9
5 LIMITATIONS.....	11

1 OVERVIEW

Virtual DataPort (VDP) 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 VDP cache engine allows two main modes: Partial and Full.

- **Partial mode:** the cache only stores some of the tuples of the view and, at runtime, when a user queries a view the server checks if the cache contains the data required to answer the query. If the cache does not have this data, the server queries the data source.
- **Full mode:** the data of the view is always retrieved from the cache database instead of from the source.
 - **Incremental mode:** the data is obtained from the cache and merged with the most recent data from the source. Data retrieval from the source is based on a condition like 'last_modified > latest_cache_refresh'

The **Denodo Incremental Cache Load Stored Procedure** also uses the incremental strategy. In this case, the stored procedure makes incremental additions with the new tuples and the updated ones to keep the cache view updated. This process should be executed periodically, using for example the Denodo Scheduler, to keep the cache up to date. Note that you have to **activate the full cache option** in the view to be cached.

The approach of this stored procedure is the following:

1. Given a view that has a primary key
2. It retrieves the identifiers from the source that do not exist yet in the cache, based on a condition configured by the user.
3. Those identifiers are used as values in the IN operator in the WHERE clause of the queries that will update the cache.

2 INSTALLATION

2.1 IMPORTING THE STORED PROCEDURE

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, using the File > Jar Management menu of the VDP Administration Tool (or File > Extension management in Denodo 7.0).

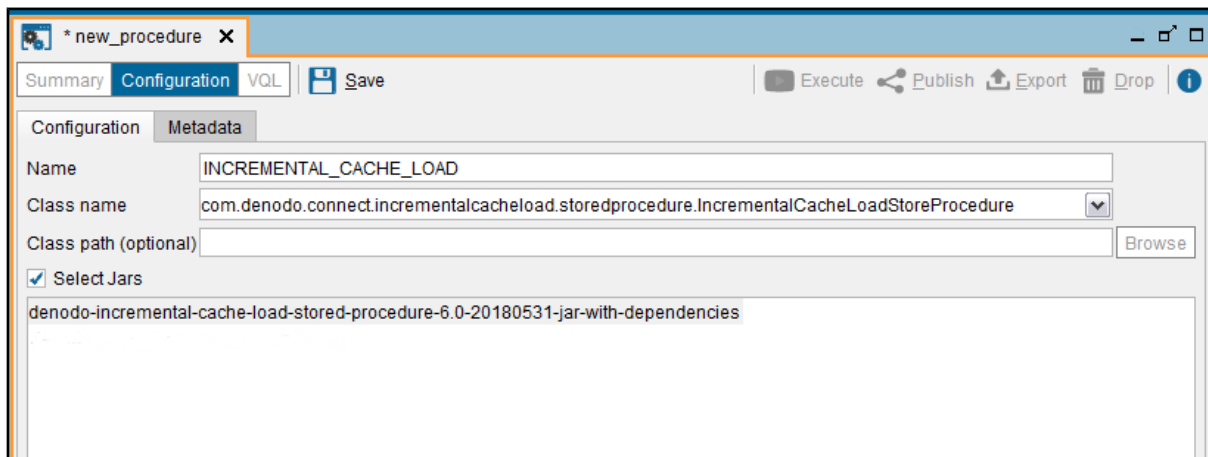
2.2 ADDING THE STORED PROCEDURE: VQL SHELL

You can create the stored procedure with the statement `CREATE PROCEDURE`:

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

2.3 ADDING THE STORED PROCEDURE: VIRTUAL DATAPORT ADMINISTRATION TOOL MENU

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



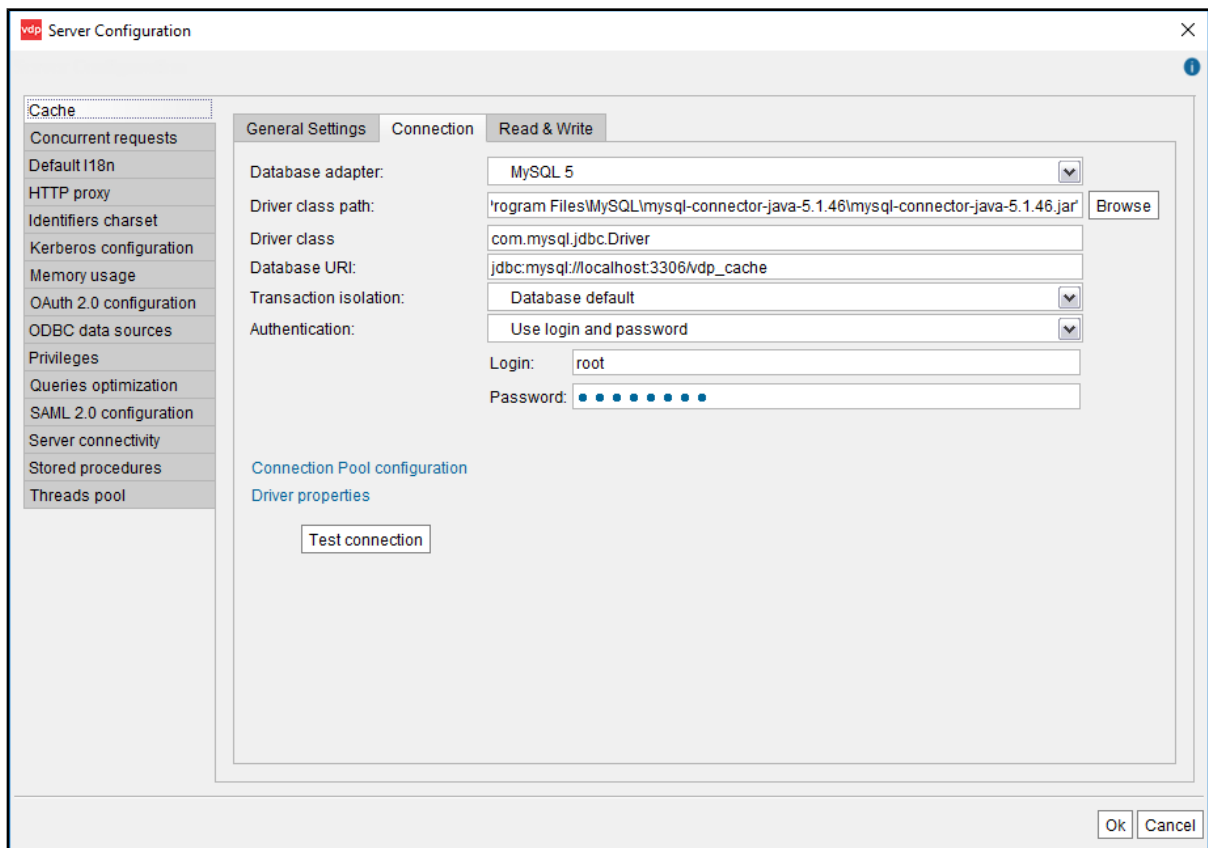
You must set a name in the Name field and select the Select Jars checkbox in order to use the Jar file, `denodo-incremental-cache-load-<vdpversion>-<version>`, previously added to the Virtual DataPort Server (see the **Importing the extension to the Virtual DataPort Server** section for more information).

3 REQUIREMENTS

3.1.1 VDP Cache module

The Virtual DataPort **cache module has to be enabled**.

When you enable the Cache the default Database Management System (DBMS) 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 configuration with MySQL 5:



The screenshot shows the 'Server Configuration' window for the 'Cache' module. The 'Connection' tab is selected, 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.

3.1.2 Primary key

The view that is going to be cached, **must have a primary key**. As the process that updates the cache uses the primary key for considering whether a row from the cache and a row from the source are the same or not.

3.1.3 Privileges required

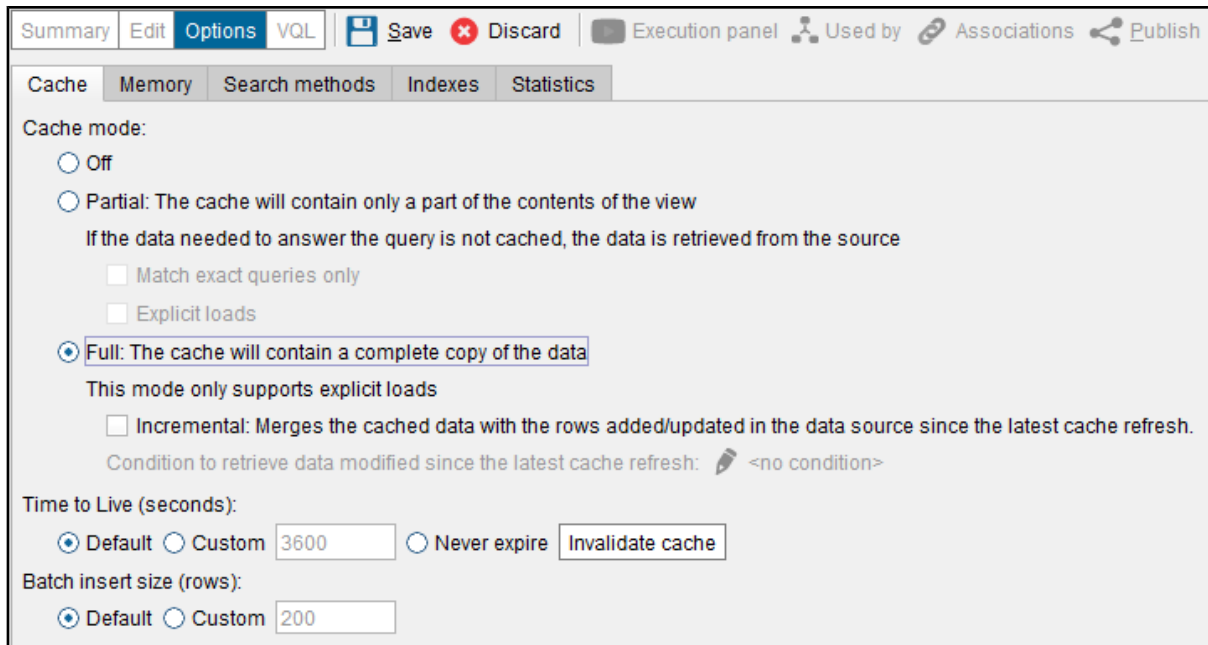
The following user privileges are required:

- CONNECT privileges over the **DATABASE_NAME** specified as the input parameter.

- READ privileges over the **VIEW_NAME** specified as the input parameter

3.1.4 View configuration

You have to activate the full cache option in the view to be cached.



The screenshot shows the 'Options' tab in the Denodo interface. The 'Cache' section is active, showing the following configuration:

- Cache mode:**
 - ☐ Off
 - ☐ Partial: The cache will contain only a part of the contents of the view. If the data needed to answer the query is not cached, the data is retrieved from the source.
 - ☐ Match exact queries only
 - ☐ Explicit loads
 - ☒ Full: The cache will contain a complete copy of the data. This mode only supports explicit loads.
 - ☐ Incremental: Merges the cached data with the rows added/updated in the data source since the latest cache refresh. Condition to retrieve data modified since the latest cache refresh:
- Time to Live (seconds):**
 - ☒ Default
 - ☐ Custom
 - ☐ Never expire
- Batch insert size (rows):**
 - ☒ Default
 - ☐ Custom

4 EXECUTION

The stored procedure requires the following input parameters:

- **DATABASE_NAME** (mandatory): the Virtual DataPort's database name where is the view that you want to cache.
- **VIEW_NAME** (mandatory): the view name that the Denodo Incremental Cache Load Stored Procedure will cache.
- **LAST_UPDATE_CONDITION** (mandatory): the condition that retrieves data from the source that do not exist yet in the cache and that will be loaded in the cache by this stored procedure.
For example: 'registered_date > ''2018-05-10'' ' (note that you have to escape the single quotes).
- **NUM_ELEMENTS_IN_CLAUSE** (mandatory): the chunk size of the IN operator in the queries that are going to update de 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 limitations.
As every scenario is different, it is difficult to propose an optimal chunk size. However, chunks between 5.000 and 20.000 get the best performance.

Note that if database or view were created with special chars or if they are case sensitive, the parameter must be surrounded with double quotes. For example, if you have a view called `bv_example.view` you should set the `VIEW_NAME` parameter as `"bv_example.view"`:

```
CALL INCREMENTAL_CACHE_LOAD('test test', '"bv_example.view"', 'id > 20', 10000);
```

There are four possibilities for executing the stored procedure:

1. Click the Execute button in the dialog that displays the schema of the stored procedure. The VDP Administration Tool will show a dialog to enter the input values.

Stored procedure parameters

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

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

Ok Cancel

2. Execute the CALL statement from the VQL Shell:

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

3. Execute as a SELECT statement in the VQL Shell:

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

4. Execute as a SELECT statement in the VQL Shell:

```
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 **has an output parameter**, NUM_UPDATED_ROWS, that returns the number of rows updated in the cache.

4.1 USE OF DENODO SCHEDULER

To perform cache loads periodically in order to keep the data updated in the cache, you can use Denodo Scheduler.

For this, you have to define a daily job with a condition like 'lastmodifieddate >= ADDMINUTE(ADDDAY(NOW(), -1), -1)' in the **LAST_UPDATE_CONDITION** parameter. Of course, you can adjust the formula to make the job run hourly, weekly, ...

4.2 USING THE SP WITH DERIVED VIEWS

It is important to notice that, due to the nature of the process performed by this stored procedure, it's quite simple to use it with **base views**, as you may only have to check an atomic condition over some data field of the mentioned view. But for **derived views**, you will have to make sure that you update the cache when you have new records in any of the views underlying the derived view.

Consider a join view with these two views: customer and support cases. And that the **LAST_UPDATE_CONDITION** is like last_modified > @LASTCACHEREFRSH.

CUSTOMER ===== c_id name last_modified ----- 1 ACME 20180626 2 EMCA 20180626	SUPPORT CASE ===== sp_id c_id description last_modified ----- 1 1 desc1 20180626 2 1 desc2 20180626
DV_CUSTOMER_J_SUPPORT CASE ===== c_id name c_last_modified sp_id description sc_last_modified ----- 1 ACME 20180626 1 desc1 20180626 1 ACME 20180626 2 desc2 20180626	

In the first execution of the stored procedure both the content of customer and support case will be loaded in the cache of this join view.

CUSTOMER ===== c_id name last_modified ----- 1 ACME 20180626 2 EMCA 20180626	SUPPORT CASE ===== sp_id c_id description last_modified ----- 1 1 desc1 20180626 2 1 desc2 20180626 3 2 desc3 20180627
DV_CUSTOMER_J_SUPPORT CASE ===== c_id name c_last_modified sp_id description sc_last_modified ----- 1 ACME 20180626 1 desc1 20180626 1 ACME 20180626 2 desc2 20180626	

1		ACME		20180626		3		desc3		20180627
---	--	------	--	----------	--	---	--	-------	--	----------

In the next execution of the stored procedure the next day, for example, the only tuple that matches the **LAST_UPDATE_CONDITION** is the one added in the support case. In the customer side there are no new tuples. If the condition set in the stored procedure only checks one of the views (like we'll usually do for base views), you may lose data in the cache.

So if we execute the stored procedure as follows:

```
CALL INCREMENTAL_CACHE_LOAD('test', 'dv_customer_j_support_case',
'c_last_modified > @LASTCACHEREFRRESH', 1);
```

The procedure won't find any rows matching the condition because there aren't new registers in the customer view and the cache won't be updated.

The most effective solution to this issue -valid even for cases where referential integrity might not be enforced- is including in the cache update condition the update conditions of all involved base views, so in this example will result this way:

```
CALL INCREMENTAL_CACHE_LOAD('test', 'dv_customer_j_support_case',
'c_last_modified > @LASTCACHEREFRRESH OR sc_last_modified >
@LASTCACHEREFRRESH', 1);
```

Note that **we need to use OR with the conditions**, otherwise we may face the same problem that we had when we only checked the last modified date of one view only. If the clause AND is used, the cache won't be updated because only `sc_last_modified > @LASTCACHEREFRRESH` is true.

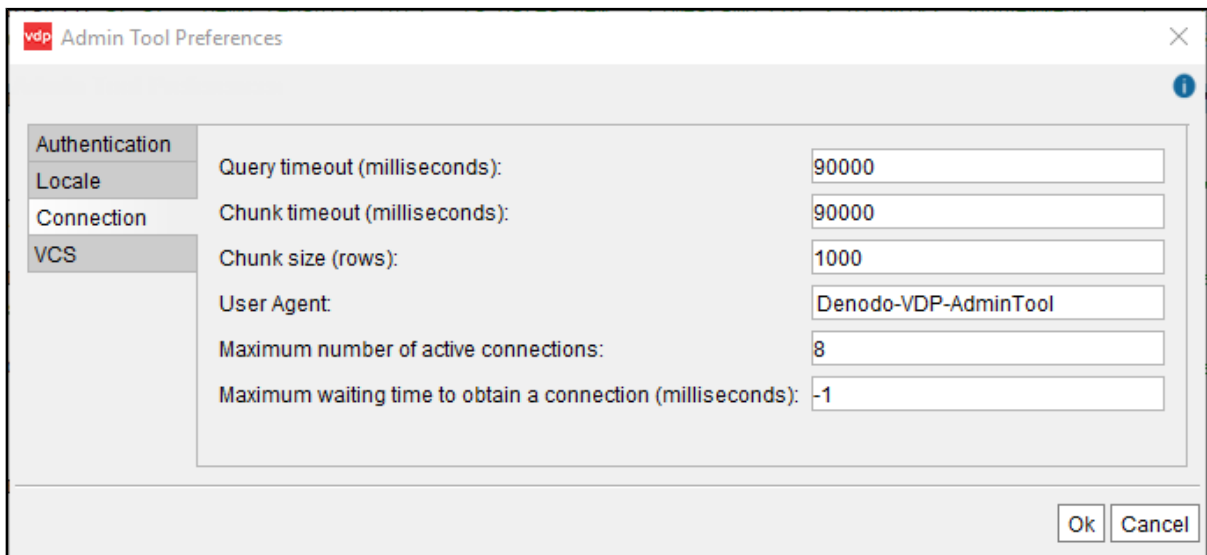
5 LIMITATIONS

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.



Tab	Property	Value
Authentication	Query timeout (milliseconds):	90000
	Chunk timeout (milliseconds):	90000
Locale	Chunk size (rows):	1000
	User Agent:	Denodo-VDP-AdminTool
Connection	Maximum number of active connections:	8
	Maximum waiting time to obtain a connection (milliseconds):	-1
VCS		

Derby limitation with CONCAT function

VDP doesn't delegate the CONCAT function over Derby data sources at the moment. Caching views with a primary key composed by more than one field needs to use this function in the scenario where any amount of rows need to be invalidated, so in the case of complex primary keys you won't be able to use Derby as cache and you'll need to use another different source.