



# Denodo Incremental Cache Load Stored Procedure - User Manual

Revision 20180627

## 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

|  |           |
|--|-----------|
| <b>1 OVERVIEW.....</b>   | <b>3</b>  |
| <b>2 INSTALLATION.....</b>   | <b>4</b>  |
| 2.1 IMPORTING THE STORED PROCEDURE.....  | 4         |
| 2.2 ADDING THE STORED PROCEDURE: VQL SHELL.....                                    | 4         |
| 2.3 ADDING THE STORED PROCEDURE: VIRTUAL DATAPORT<br>ADMINISTRATION TOOL MENU..... | 4         |
| <b>3 REQUIREMENTS.....</b>   | <b>5</b>  |
| <b>4 EXECUTION.....</b>  | <b>7</b>  |
| 4.1 USE OF @LASTCACHEREFFRESH VARIABLE.....  | 9         |
| 4.2 USE OF DENODO SCHEDULER.....   | 9         |
| <b>5 LIMITATIONS.....</b>  | <b>10</b> |

## 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.

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).

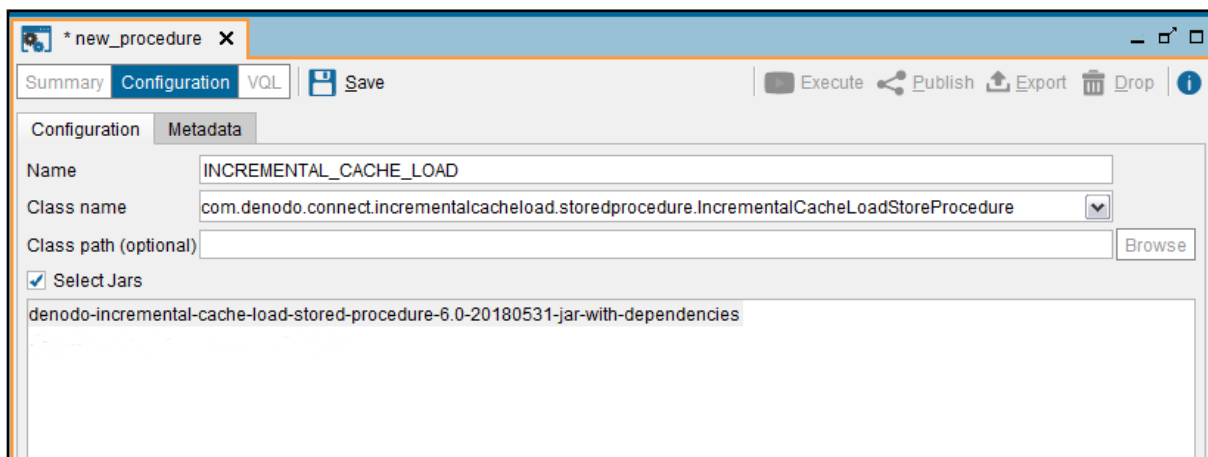
### 2.2 ADDING THE STORED PROCEDURE: VOL SHELL

You can create the 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>';
  [ 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:



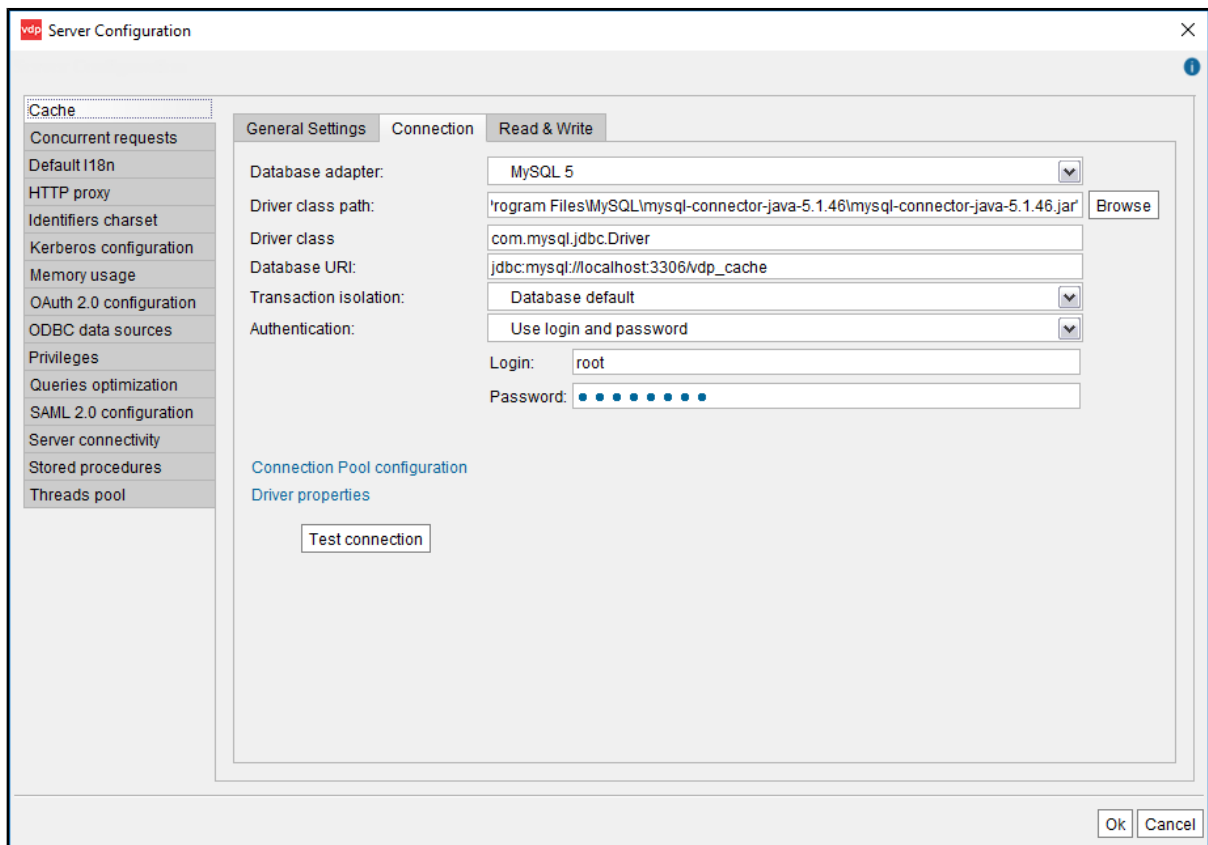
You must set a name in the Name field and select the Select Jars checkbox in order to use the Jar file, `denodo-incremental-cache-load-<vdpversion>-<version>`, previously added to the Virtual DataPort Server (see the **Importing the extension to the Virtual DataPort Server** section for more information).

## 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:



### 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 Base views and Derived views

It is important to notice that, due to the nature of the process performed by this stored procedure, it **works well with base views**. But it has some **limitations for derived views**.

As the primary key and the update condition are the basis of this stored procedure, Derived views, for example: Projections, must contain the primary key and the columns involved in the update condition (**LAST\_UPDATE\_CONDITION**)

Next, follows an example of a Join view that does not behave well with the **Denodo Incremental Cache Load Stored Procedure**:

| CUSTOMER<br>===== |      |               | SUPPORT CASE<br>===== |      |             |               |
|-------------------|------|---------------|-----------------------|------|-------------|---------------|
| c_id              | name | last_modified | sp_id                 | c_id | description | last_modified |
| 1                 | ACME | 20180626      | 1                     | 1    | desc1       | 20180626      |
| 2                 | EMCA | 20180626      | 2                     | 1    | desc2       | 20180626      |

Consider a Join view with these two views: customer and support cases. And that the **LAST\_UPDATE\_CONDITION** is like `last_modified > @LASTCACHEREFRESH`.

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<br>===== |      |               | SUPPORT CASE<br>===== |      |             |               |
|-------------------|------|---------------|-----------------------|------|-------------|---------------|
| c_id              | name | last_modified | sp_id                 | c_id | description | last_modified |
| 1                 | ACME | 20180626      | 1                     | 1    | desc1       | 20180626      |
| 2                 | EMCA | 20180626      | 2                     | 1    | desc2       | 20180626      |
|                   |      |               | 3                     | 2    | desc3       | 20180627      |

In the next execution of the stored procedure the next day, for example, the only tuple that matches with the **LAST\_UPDATE\_CONDITION** is the one added in the support case. In the customer side there are no new tuples. So the join will return no results and the cache won't be updated.

### 3.1.4 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

## 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.

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="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

- Execute the CALL statement from the VQL Shell:

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

- Execute as a SELECT statement in the VQL Shell:

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

- 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 @LASTCACHEREFFRESH VARIABLE

For building the **LAST\_UPDATE\_CONDITION**, the condition that selects the tuples to be updated in the cache, you can use the `@LASTCACHEREFFRESH` variable. This variable allows you to create conditions based on the last cache refresh.

To be able to use this variable, the view to be cached has to have a field that contains when a row was last inserted/updated. The condition will be of the form: `'modified_date > @LASTCACHEREFFRESH'`.

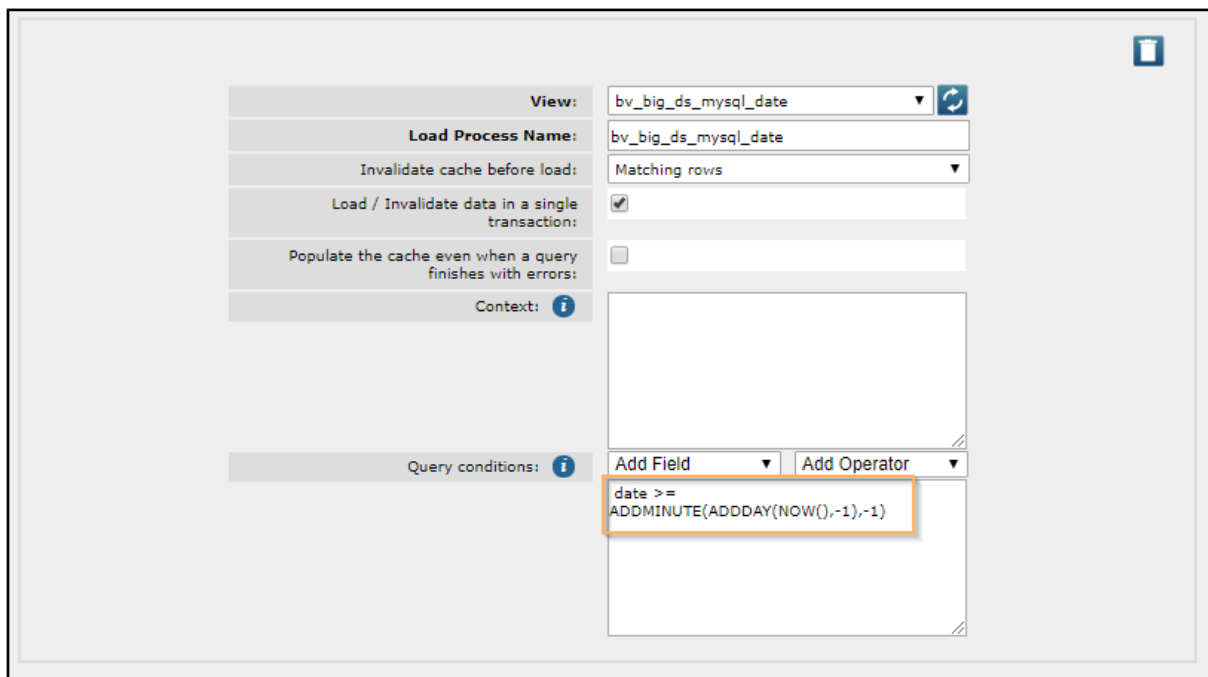
```
CALL incremental_cache_load('database_name','view_name',
    'modified_date > @LASTCACHEREFFRESH','1000');
```

#### 4.2 USE OF DENODO SCHEDULER

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

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, ...

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



The screenshot shows the Denodo Scheduler configuration page for a new job. The 'View' dropdown is set to 'bv\_big\_ds\_mysql\_date'. The 'Load Process Name' is also 'bv\_big\_ds\_mysql\_date'. The 'Invalidate cache before load' dropdown is set to 'Matching rows'. The 'Load / Invalidate data in a single transaction' checkbox is checked. The 'Populate the cache even when a query finishes with errors' checkbox is unchecked. The 'Context' field is empty. The 'Query conditions' section shows a condition: 'date >= ADDMINUTE(ADDDAY(NOW(),-1),-1)'. The 'Add Field' and 'Add Operator' buttons are visible above the condition input.

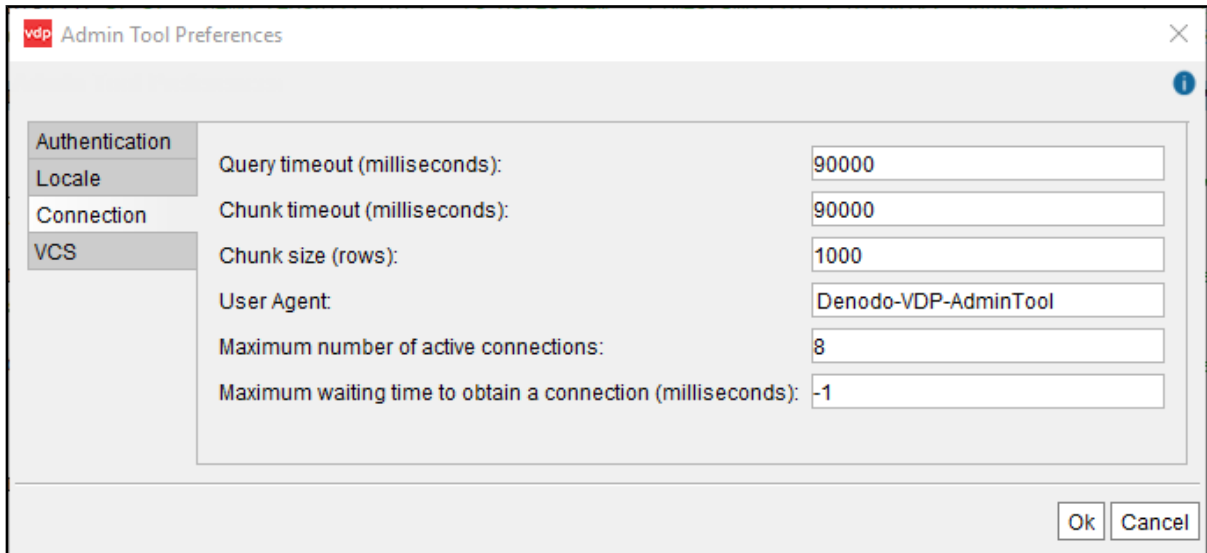
## 5 LIMITATIONS

### 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.



| Category       | 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            |   |                      |