Denodo Admin and Development Best Practices

Revision 20230907

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

CONTENTS

1 INSTALLATION

1.1 WINDOWS INSTALLATION

The Denodo Platform installs under a single folder (DENODO_HOME) and writes data in the same folder hierarchy so to avoid issues with permissions the recommended installation path should be outside the `C:\Program Files` or `C:\Program Files x86` folder.

Modern Microsoft Windows systems (Vista, 2008, 7, 8, 10...) are protecting the `C:\Program Files` folder so that applications can not write in it. Windows recommends using the `C:\Program Data` or `%APPDATA%` folders (the latter is a sub-folder of the user home) for this purpose. Current versions of the Denodo Platform are not using these folders, but storing the metadata in the same program installation hierarchy.

We could use, for example, the `C:\Denodo` as the base installation directory. In this case the Denodo Platform should be located in the `C:\Denodo\DenodoPlatformx.y` folder: for example, `C:\Denodo\DenodoPlatform8.0`. If we need to develop other non-Denodo components we could also use the `C:\Denodo` folder or we could use that folder to put project documentation or data files.

1.2 LINUX INSTALLATION

The recommended folder for Linux installations is `/opt/denodo` as the base folder, so the Denodo Platform folder should be `/opt/denodo/denodo_platform_x.y`, for example: `/opt/denodo/denodo_platform_8.0`.

Depending on the load the system will support, it might become necessary to increase the maximum allowed number of open files. This parameter is directly affected by the maximum number of threads configured, and can be changed in an `/etc/security/limits.conf` (though the specific path may depend on the Linux distribution). The current limit established for a running process can be checked at `/proc/{pid}/limits`.

2 ADMINISTRATION

2.1 DATABASE MANAGEMENT

A Virtual DataPort database is equivalent to a virtual schema, and therefore is the unit of project work in the Denodo Platform. The following suggestions apply regarding database management:

2.1.1 Development

- Create a new database for each new application/project using the Platform.
- Create new databases for each user in a project that might need to perform frequent modifications on the project's database. Create these databases as replicas of the project's trunk and merge changes when considered stable.
- Use VQL export/import tools for replicating existing databases and performing version control of your schemas.

2.1.2 Production

- Do not mix development and production schemas in the same server.
- Test new modifications in a different schema in the same server before applying these modifications to the production schema. If a dedicated pre-production server is available, use it for these purposes.
- Replicating views among diverse virtual schemas is acceptable if this makes sense for your application needs. Give applications/clients what they need and no more.
- Always apply source control tags to the production versions of your virtual schemas' metadata.

2.2 SECURITY: ROLES, USERS, PERMISSIONS

Denodo Platform allows you to define a complete security infrastructure in which each user can be assigned specific permissions on each of your databases, and these permissions can be grouped into roles for easier maintenance. Roles can also be hierarchical.

Using roles is, in fact, the recommended security setup for most installations.

Denodo distinguishes between two types of users:

- **Administrators**. These can create, modify and delete databases in a DataPort server without any limitation. Likewise, they can also create, modify and delete users. When the server is installed, a default administrator user is created whose name is admin and whose password is also admin. This user can n
- ever be deleted.
- **Normal users**. These cannot create, modify or delete users. They cannot create or delete databases, although they can have connection, read, create or write privileges to one or several databases or to specific views contained therein.

The first step to be taken by the administrator should be the change of the password of

the default user. More detail on how to configure the server can be found in the VDP Administration Guide, section Administration of databases, users, roles and their access rights.

It is also recommended to create normal user accounts for the developers using the tool. LDAP or Active Directory users and groups can be configured and used if the organization uses such systems for authentication. For more information on how to manage users, please refer to the Administration Manual, section User and access rights in Virtual DataPort.

It's also a good practice to set up a specific user for each consuming application. This user should be a "normal user" with privileges only for the views that are going to be used by the application. This way, all the auxiliary views and data sources will be hidden, offering an interface for the application developers completely independent from the original sources, and easier to understand.

2.3 JVM CONFIGURATION

Denodo Virtual DataPort server is a java application running on a Java Virtual Machine. As such, its launch scripts can be modified to specify parameters that could help improve performance in a particular scenario.

Such launch scripts are located inside the platform installation folder, under the <DENODO_HOME>/bin/ folder. In particular, the launch script for the Virtual DataPort server is vqlserver.bat in Windows environments or vqlserver.sh for Unix systems.

By default, this script specifies the following JVM option among others:

- `-Xmx4096m`: maximum size of JVM heap, equivalent to the maximum amount of memory the JVM will be allowed to use. A value of 4Gbytew is here expressed as `4096m`.

This configuration parameter can be easily changed in the Denodo Control Center ("`Configure`" > "`JVM Options`").

If the Denodo Platform is installed on an environment without graphical interface, this parameter can be modified by editing the configuration file of Virtual DataPort (`<DENODO_HOME>/conf/vdp/VDBConfiguration.properties`) in the following property:

```
java.env.DENODO_OPTS_START
```

Once the configuration file has been updated, you have to execute the `<DENODO_HOME>/bin/regenerateFiles.sh` script in order to propagate the changes in the startup scripts. After the execution of this script, the next time a server or a tool is started, it will use the new JVM startup options.
Starting from Denodo 7.0 this configuration can be changed using the SET command from a VQL shell.

Note: these modifications are finally applied to the startup scripts (`vqlserver.bat` or `vqlserver.sh`), but these files should never be modified directly. Installing a new update can modify these files automatically and therefore the startup options could be overwritten.

2.3.1 Recommended settings for the JVM when Denodo Platform needs to deal with huge datasets

If Denodo needs to deal with huge datasets, it might be needed to increase the default heap size (which is 4096Mb). If with this heap size, swapping is still needed for some queries, and there is available memory in the machine, the administrator can try to increase it.

In order to determine the maximum amount of memory we can assign, we have to check the amount of physical memory available before launching Denodo Virtual DataPort (VDP). Do not assign all the available memory to VDP. Leave at least around 300 / 400 Mb free. Think about whether there will be more processes running on the machine at the same time as VDP, and if they will use more memory in the future.

Some Important Points to Remember:

- Do not assign more memory than what it is available. Otherwise, the OS will have to swap the memory of other processes to disk and the performance of the machine and all its processes will decrease.
- If the server is running in a virtual machine, try to guarantee that the memory assigned to that virtual machine is actually reserved for it (that is, the allocated memory cannot be used by other virtual machines, even when VDP is not using it). If this is not possible, be conservative in setting the size of the allocated heap size to avoid the OS swapping to disk.
- If we assign less than 1.4 Gb to VDP's heap, we must use a 32-bit JRE. By using a 64 bits, performance will be impacted, as every object will cost twice as much to address. Use a 64-bit JVM only if the size of the heap is 1.4 Gb or bigger.

Starting from Denodo 7.0, the recommended settings (the following section) for the JVM can be directly set from the VQL shell:

```
SET 'java.env.DENODO_OPTS_START' = '<JVM settings>';
```

When changing the memory configuration with SET, VDP automatically invokes the `<DENODO_HOME>/bin/regenerateFiles` script. If you modify the file manually, you have to remember to do it. Note that, as this is a change in the JVM configuration, the VDP server will still need to be restarted after setting new values.

For more details and examples refer to Changing Settings of Virtual DataPort and the Web Container.

2.3.2 Recommended settings for the JVM in memory-demanding scenarios

Note that for a batch queries scenario where pauses caused by the garbage collector do not represent a problem for the functionality of the system, it is not recommended to use a heap size greater than 16 Gb. This configuration assigns 8 Gb to the JVM's heap:

For Denodo 7.0 and 8.0:

```
-server -Xms8192m -Xmx8192m -XX:+DisableExplicitGC -XX:+UseG1GC
-XX:ReservedCodeCacheSize=256m
```

For Denodo versions prior to 7.0, the following recommendation can be used to assign 8 Gb to the JVM's heap:

```
-server -Xms8192m -Xmx8192m -XX:MaxPermSize=256m -XX:
+DisableExplicitGC -XX:+UseG1GC -XX:ReservedCodeCacheSize=256m
```

2.3.3 Recommended settings for the JVM in real time scenarios

For a scenario where real time queries are being executed it is not recommended to use a heap size greater than 8 Gb. With larger heap sizes the execution of the garbage collector can last a long time (even minutes) and it will affect the response time when the garbage collector is running if the heap size is larger than 8 Gb. For very quick queries returning only a few rows the following configuration is recommended:

For Denodo 7.0 and 8.0:

```
-server -Xms4096m -Xmx4096m -XX:+DisableExplicitGC -XX:+UseG1GC
-XX:MaxGCPauseMillis=1000 -XX:ReservedCodeCacheSize=256m
```

Use the following configuration for Denodo versions prior to 7.0:

```
-server -Xms4096m -Xmx4096m -XX:MaxPermSize=256m -XX:
+DisableExplicitGC -XX:+UseG1GC -XX:InitiatingHeapOccupancyPercent=60
-XX:ReservedCodeCacheSize=256m
```

or

```
-server -Xms4096m -Xmx4096m -XX:MaxPermSize=256m -XX:
+DisableExplicitGC -XX:+UseG1GC -XX:MaxGCPauseMillis=1000
-XX:ReservedCodeCacheSize=256m
```

The option "`-XX:InitiatingHeapOccupancyPercent=60`" can be used in scenarios where query load is very variable and load peaks are expected as it allows to keep the heap space with less use for these peaks.
The option "`-XX:MaxGCPauseMillis=1000`" can be used in scenarios where the query load is uniform to guarantee consistent times in garbage collector pauses. This option sets the maximum pause time for the garbage collector.

Explanation of these settings:

- **`-server:`** with this option, the JVM and its Garbage Collector (GC) are configured to run server applications. That means that the default values of the JVM and its Garbage Collector (GC) will be oriented to execute server-class applications. Server-class applications have different needs than client applications. For example, client applications need faster startup times than server ones even if in the end, the throughput is slightly worse.
- **`-Xms`** and **`-Xmx`**
  - `-Xms`: amount of allocated memory. Allocated memory is fully backed by physical memory.

  - `-Xmx`: amount of reserved memory. The JVM announces to the OS that at some point, it may request this amount of memory, but it does not need it yet. The reserved memory can be used by other processes while is not allocated by the JVM.
  - Both parameters should have the same value in server-class applications. When the JVM is launched, it allocates a region of memory with size `-Xms` and, if needed, it requests more memory to the operating system (OS), until it reaches the `-Xmx` limit. Allocating more memory for the JVM is a rather expensive operation because the JVM memory has to be contiguous and if there is not any contiguous space, the OS has to swap the memory used by other processes until it obtains a region with the requested size. This process can even lead to swapping the memory of some processes to disk.
  - In server-class applications it is a good idea to put the same value to `-Xms` and `-Xmx` because eventually, the memory used by the JVM will grow to the top size (`-Xmx`). The boot process may be slightly slower (we are talking about a difference in seconds) because it has to allocate more memory, but in the end it will result in a performance gain. The reason is that DataPort will not have to request any more memory to the O.S during the execution.
- **`-XX:MaxPermSize:`** Top size of the Permanent Generation of the JVM's heap. Note that it is not included on or after Denodo 7.0, since this property is no longer supported in Java 8 and 11.
- **`-XX:+DisableExplicitGC:`** The JVM ignores all invocations to `System.gc()` from the code. The JVM still performs garbage collection when necessary.
  It is usually a good idea to enable this option because a developer of an external jar (i.e.: custom wrapper, stored procedure, ...) loaded into Virtual DataPort can invoke a `System.gc()` to free some memory. However, this may force the GC to start collecting garbage even when there is enough free space in the heap. This can even lead to serious performance issues, especially if Virtual DataPort is receiving many simultaneous requests.
- **`-XX:ReservedCodeCacheSize:`** This parameter defines the size of the region of the heap in which the JVM stores compiled native code.
- **`-XX:+UseG1GC`**: tells the JVM to use the G1 Garbage collector.
- **`-XX:MaxGCPauseMillis`**: sets a target for the maximum GC pause time. This is a soft goal, and the JVM will make its best effort to achieve it.
- **`-XX:InitiatingHeapOccupancyPercent:`** This parameter sets the Java heap occupancy threshold that triggers a marking cycle.

Note: These notes are specific to the Sun/Oracle JVM. Differences in configuration when using other JVMs such as IBM's or jRockit might be needed.

2.4 SYSTEM MEMORY MANAGEMENT

In some cases, the memory used by the Java process running the Virtual DataPort server can be quite high and higher than the maximum heap memory configured. This is not necessarily an issue as the Java Virtual Machine needs more memory in addition to the heap memory for internal management.
In Linux systems, under some circumstances, this off-heap memory usage keeps growing and it might reach levels that are not desirable. This is caused by the `malloc` (memory allocation) operation implemented in the `glibc` library used by most Linux

operating systems where a performance enhancement might end up reserving more memory than needed. By default, `glibc` uses a total amount of `8*number_of_cores` memory pools when reserving memory and this consumes too much space in the JVM. To minimize the memory usage it is possible to set an environment variable in the system called `MALLOC_ARENA_MAX`.

```
export MALLOC_ARENA_MAX=<2 x (# ofCPU Cores)>
```

In our tests a value of 2 by the number of cores in the system did not cause any performance penalty and significantly decreased the memory usage. A constant value of 4 can also be used to lower the memory usage even more if needed.

2.5 REPLICATION

Virtual DataPort can operate within a replicated architecture to provide diverse levels of high availability via load balancing. Typically, load balancing architectures for DataPort operate as follows:

1. An external load balancer system distributes the requests received among a pool of Virtual DataPort servers.
2. The load balancer system detects when a server is not responding (using the so-called *'server health check'*). In this case, the load is distributed among the remaining servers.
3. The clustering system detects when the server is available again (using the 'health check') and adds it to the pool.

Denodo provides a ping script for the server health check operation. Depending on its configuration, the ping script can just check that the server is alive, or it can execute a simple query (for example "`select * from dual()`"). The script, as well as more detailed information on how to use it, can be found in the file denodo-db-tools.tar.gz or denodo-db-tools.zip located in the installation folder, under tools/db.

In order to facilitate metadata replication among the servers in a cluster, Denodo provides a set of tools to manage those operations from the command line. Besides, the export script can receive the list of servers from the cluster, replicating the metadata in all the servers with just one command. The scripts, as well as its usage details, can be found in the file denodo-db-tools.tar.gz or denodo-db-tools.zip located in the installation folder, under tools/db.

More information on how to set up a load-balancing environment and its utilities can be found in the Administration Guide, section Cluster Architectures / Server backup.

2.6 CACHE CONFIGURATION

Virtual DataPort has a cache system to store local copies of the source data as required. This cache is stored in a Relational Database accessible through JDBC. DataPort embeds an Apache Derby database which can be used to store the cache information. It is also possible, and recommended for production scenarios, to use the following external RDBMSs to store the cache: MySQL, Oracle or MS SQL Server.

The system will generate and automatically maintain a table in the cache Database for each base relation or view for which the cache has been enabled.

Each view in the cache has an associated expiration time. Once tuples have expired, they are not considered in queries and are automatically deleted at regular intervals. This mechanism also allows data to be preloaded periodically in a very simple fashion by simply writing a query describing the data to be preloaded, and scheduling the query to be repeated at the desired time interval. The cache can also be disabled for a specific query or views.

More information on how to set up the cache can be found in the Administration Guide, sections Configuring the Cache and Cache Module, and in the section Configuring the cache of a view.

2.6.1 Sharing the cache in a cluster

The architecture of the cache makes it perfect to be used as a shared mechanism among a cluster of servers. If as recommended, an external RDBMS is used as storage for the cache, simply loading the VQL file pointing to that server will set up the use of a shared cache.

2.6.2 Consideration on the use of the cache

The cache system is an RDBMS, and its access is done via a connection pool using JDBC. Thus, the connection pool parameters must be configured accordingly. An overuse of the cache may end up making the system slower, if the waiting times in the cache connection pool are slower than actually going to the original source.
Also, since the cache RDBMS will have an intensive insert/delete use, its tables should be compacted in accordance to avoid defragmentation of its data files.

2.7 MONITORING AND LOGGING

2.7.1 Logs

Denodo uses Apache Log4j 2 logging mechanism to generate the log files for its servers. The logs for the Virtual DataPort server are located in the installation folder, under `logs/vdp.log`. This is the first place to look when an error takes place in the server.
The log4j configuration file can be modified if needed, changing the logging levels or the max historical data that is stored, modifying the file `conf/vdp/log4j2.xml`

2.7.2 JMX

It is possible to access server monitoring information using the Java Management extensions (JMX) standard. This information can be used to control the use of the server and audit the actions carried out on the data sources and/or the Virtual DataPort metadata.

The available monitoring information is:

- General information on the server: number of active and inactive connections, memory usage, etc.
- Configuration and access to the server cache.
- Access to each data source.
- Access to the views and stored procedures.

- Transactions executed by the Virtual DataPort server.

Virtual DataPort also generates different types of JMX notifications whenever:

- A DDL statement is executed. E.g. `ALTER TABLE`, `CREATE VIEW`, etc.
- A DML statement is executed. E.g. `SELECT`, `INSERT`, etc.
- Transactions start or end.

External monitoring applications using JMX like HP Open View, NAGIOS, Cacti, etc can be set up to monitor the state of the server in real time.

Detailed information on the information available using JMX can be found in the Virtual DataPort Administration Guide, section Monitoring with a Java Management Extensions (JMX) Agent.

2.7.3 Diagnostic & Monitoring Tool

The Diagnostic & Monitoring Tool is a web tool included in the Denodo Platform for versions after 6.0 that provides two main features:

- Monitor the current state of a Virtual DataPort server or an environment (group of servers).
- Analyze its state in the past in order to identify the cause of a problem.

You can get more information regarding this tool in the Diagnostic / Monitoring Tool Guide.

2.7.4 Denodo Monitor

The Denodo Monitor is a tool included in the Denodo Platform that logs several parameters of the Denodo servers. Denodo monitor includes three types of “monitors”

- Local monitors: They gather information about the machine where the Denodo Monitor is running.
- Server monitors: They obtain monitoring information by connecting to a server via JMX. Thus, they can obtain this information from remote servers.
- VDP queries monitor. It logs several parameters of all the VQL queries by a VDP server. It also obtains this information via JMX, so it can monitor remote servers.

The Denodo Monitor is located in the directory `$DENODO_HOME/tools/monitor`, compressed in a zip file. Detailed information on the information available using the Denodo Monitor can be found in the Denodo Monitor section of the VDP Administration Guide.

2.7.5 Query Monitor

The Query Monitor is a tool that lists the queries that the Virtual DataPort server is currently executing. If we select a row of the table, the tool displays the tree view of the query, the tree of views used to form the results of the query. Clicking on each node of the tree displays information about it, such as number of processed rows, state, etc.

2.8 DENODO IN VIRTUAL MACHINES

Many factors can influence performance in VMs, as physical infrastructure, memory and processors overcommitment, virtual network interface or virtual disk access. These factors may depend on VM Vendor and version, Host OS or Guest OS.

2.8.1 Recommendations to stress tests on Virtual Machines

First, you should run a small subset of tests in a physical machine (e.g. 4 cores) to obtain a baseline for comparison. After you should run the same tests in VM of similar theoretical power, for instance if you use VMWare, you can tune the configuration to get similar performance, setting the memory assignment, the processor assignment, the monitor virtual network adapters and disk access performance, etc. Finally you should run all tests using scaled up or scaled out VM, if it is required.

2.8.2 Environment-dependent factors

There are environment-dependent factors which can heavily impact the performance of VMs. You should take into consideration the following issues to improve the performance of Denodo on a Virtual Machine.

2.8.2.1 Processors overcommitment

If you set VM with processor overcommitments the results will be significantly worse than reserving an equivalent number of cores, obtaining greater average(>15%) and greater standard deviation. This depends on the number of unavailable cores. Let’s say a physical architecture with 4 cores,

- with 1 core unavailable results aren't affected significantly,
- with 2 cores unavailable results are slightly affected in some tests,
- with more than 2 cores unavailable, results are significantly affected in some tests, and heavily affected in more complex tests.

Different tools may react differently to processors overcommitment according to several design factors. Denodo accesses data sources in parallel using different threads. It leverages modern multi-core architectures, due to data source access is an usual bottleneck. But this strategy, that is good in other situations, can be more affected by strong processors overcommitting .

The recommendation is: avoid excessive processors overcommitting.

2.8.2.2 Memory Overcommitment

It interacts badly with the Java garbage collector, for example VMWare explicitly recommends avoiding memory overcommitment with JAVA applications.

2.8.2.3 Memory assigned

You should notice the difference between memory assigned(upper limit), and reserved in the virtualization software. Even when JVM specifies memory reserved equal to memory assigned, this memory will not be reclaimed by virtual machine software until it is needed, so performance is severely affected at the beginning, but it improves as new queries are executed and VM reclaims the maximum configured memory.
For example, VMWare explicitly recommends reserving all memory for Java applications. So, the recommendation is to set the assigned memory (upper limit) equal to the reserved memory in the VM.

2.8.2.4 Virtual Network and Virtual Disks Adapters

The misconfiguration of these adapters (e.g. wrong MTU in network adapter) may have a strong impact on performance. In the case that you are using VMWare, you can use VMWare diagnostic tools to detect these problems.

3 DEVELOPMENT

3.1 DEVELOPMENT ENVIRONMENT LAYOUT

For typical configurations of a development environment, see the section Scenarios and Recommended Uses of the Virtual DataPort Administration Guide.

3.2 VIRTUAL SCHEMA NAMING CONVENTIONS

See the document VDP Naming Conventions for some useful rules on naming your views in a virtual schema:

3.3 ACCESSING VIRTUAL VIEWS FROM APPLICATIONS

Directly giving client applications access to base views partly dissolves the logical independence advantages offered by data virtualization, because any change in the sources would directly affect applications.
So it is recommended not to give applications direct access to base views, rather using derived views that mirror them for this.

3.4 DOCUMENTING VIRTUAL SCHEMAS (VIEWS)

Along with a good naming convention, views can be documented by adding a description to them. A good description may include the purpose of the view, the creator, the use case it is used for, etc... The description may be included during the generation of the view, or later in the Advanced section of the view.

By using the Catalog Search (see the VDP Administration Guide, section Catalog Search), you can locate elements based on the following criteria:

- Element Type. If the selected element type is ‘Views’, the user can also select ‘View Type’, ‘Swap active’ and ‘Cache Status’ as search criteria.
- Owner. User name of the user that created the elements.
- Last User Modifier. User name of the last user that modified the elements.
- Initial Creation Date and End Creation Date. Elements that were created between the ‘Initial Creation Date’ and the ‘End Creation Date’.
- Initial Modification Date and End Modification Date. Elements that were modified between the ‘Initial Modification Date’ and the ‘End Modification Date’.
- Description. Elements that contain the value of this field in its description.

3.5 METADATA STORAGE, VERSIONING AND SHARING

Virtual Schema definitions (the so-called *“metadata”*) can be easily represented as a sequence of VQL sentences that defines users, databases, data sources, base and derived views, web service definitions, etc.

VQL has the advantage of being a plain text format, which allows an easy storage of virtual schema definitions in VQL files, as well as versioning and applying diff tools. Also, virtual schemas can be broken into parts that can be shared and recombined to create new virtual schemas featuring structures from other schemas.

When dealing with metadata in VQL format, you should always think of two types of metadata code:

1. Environment-dependent metadata. This is metadata (or fragments of it) containing data that might change depending on the environment the software is being executed in. For example, a JDBC data source for the development instance of a database will most certainly not be used in production, where it should be substituted by the equivalent production instance of the same database. The configuration details of that JDBC data source are, therefore, environment-dependent.
2. Non-environment-dependent metadata. This is metadata (or fragments of it) that should not change at all depending on the execution environment. For example, a derived view combining data from other views should not change its definition from one environment to another (as far as the base view names do not change), so the definition of this derived view is non-environment-dependent.

Environment-dependent metadata requires special care:

- It could contain data source configuration items (users, passwords, database configurations) that might not be suitable for storage for security reasons. These pieces of data should be substituted by placeholders after exporting and before sharing the VQL.
- Placeholders for environment-dependent configurations should be correctly resolved before importing VQL again into servers (whichever the environment).

This allows us to extract a couple of important rules:

1. Virtual Schema metadata should be exported to VQL files in order to be stored and included in version control systems.
2. Stored VQL files should contain no security data. All sensitive parameters should be substituted by placeholders before being stored or shared. Typical elements to be hidden include:
    - Database URI
    - Database credentials (user/password)
    - Connection pool configurations
    - Proxy settings
    - WSDL locations and endpoints
    - Local files path

For easier maintenance, Denodo allows export the environment specific properties separately:

More details about the VQL syntax can be found in the Virtual DataPort VQL Guide.

3.6 QUERY PROFILING: DATA LINEAGE AND EXECUTION TRACES

3.6.1 Data Lineage

Data Lineage allows data admins to trace the origin of any column in a derived view, across its tree of view combinations. For any attribute of a view, VDP shows a tree view highlighting the nodes that take part of the creation of that attribute, so that you can easily know which source or sources are in fact being queried (and in which fashion) in

order to obtain any specific high-level piece of data.

3.6.2 Execution traces

After executing a statement, it is possible to access a trace of its execution. The statement execution plan is displayed in a tree diagram, where each node represents either an intermediate view involved in the execution of the statement or an access to a data source via a wrapper.

The execution trace is a very useful tool to debug and optimize the execution of queries. It shows the details of the execution in every step and branch, and makes it possible to examine the exact queries reaching the sources, as well as identify slow parts in the execution. In the event of an error, the execution trace will also highlight the branches producing such errors. More information about the execution trace can be found in the VDP Administrator Guide, section Execution Trace of a Statement.

3.7 ECLIPSE PLUGIN: DENODO4E

The Denodo4E Eclipse plug-in provides the necessary tools to create, debug and deploy Denodo applications' extensions with Eclipse. This way, extensions for Denodo applications such as VDP Stored Procedures, Custom Wrapper or Custom Functions can be developed much faster. The instructions to install the Denodo4E plugin can be found in the installation folder, in `tools/denodo4e/README`.

A new type of project, along with some new options to launch a Denodo environment will be available in Eclipse after its installation. More information on how to use the Denodo4E plug-in can be found in the Eclipse Help section (Help > Help Contents) once the plug-in is installed, under the section "Denodo4E User Guide".

Note: The Denodo4Eclipse plugin is deprecated in Denodo 8.0 and it may be removed in future major versions of the Denodo Platform.

3.8 RETURNING LARGE AMOUNTS OF DATA

When a Denodo use case is expected to return large amounts of data from one single query (in the order of thousands of tuples), the JDBC interface should be preferred when possible over Web Services publishing.

Because of its streaming nature, JDBC interfaces tend to be more efficient in this kind of scenario. SOAP/REST Web Services are built on top of HTTP (a protocol designed to transport documents, and not large amounts of data), and whereas Web Services can be highly optimized to meet high memory-usage standards, the Java JDBC API was designed with this idea from the ground up and will therefore keep an advantage in most cases.

3.9 DEALING WITH LARGE AMOUNTS OF TEXT IN CUSTOM ARTIFACTS

When developing custom artifacts -- such as custom wrappers, stored procedures or custom functions -- that process large amounts of text, Java String objects have to be handled with special care, especially if substrings are created from them.

In Java, when a substring is created from a String, the memory space allocated for the whole String remains allocated even if the substring is only a small fraction of the original String, and this memory space will never be disposed of by the garbage

collector until the substring itself also is (because the substring in fact points to the same original memory space of the original String).
This means that a wrapper that might be processing large amounts of Strings coming from lines of a text file, and creating substrings for storing chunks out of those lines, might actually be storing the complete text file in memory due to this JVM behavior.

This is avoided by creating new String objects for substrings, like "`final String chunk = new String(line.substring(0, pos));`"

3.10 DEVELOPING CUSTOM ELEMENTS USING SPRING FRAMEWORK

We can use Spring framework when developing a custom element (for example a Denodo Scheduler Custom Exporter).
Steps:

1. First, you have to copy spring libraries to `%DENODO_HOME%/extensions/thirdparty/lib` and restart the server/s.
2. Inside your project, you have to create the XML file to define the application context. It has to be in the build path of the project (in other words, it has to be inside the `my_exporter.jar` file).
   NOTE: another option, copy manually `my_exporter.xml` to `%DENODO_HOME%/extensions/dev/target/classes`
   IMPORTANT: `my_exporter.xml` has to be unique, be careful naming xml's
3. In your code, you have to initialize spring:

```
public class MyExporter implements Exporter {
      private ApplicationContext applicationContext =
            new ClassPathXmlApplicationContext(
                  new String[]{"my_exporter.xml"});
      ...
```

4. Accessing beans:

```
protected MyService myService =
      (MyService)applicationContext.getBean("myService");
...
```