NOTE

This document is confidential and is the property of denodo technologies (hereinafter denodo).
No part of the document may be copied, photographed, transmitted electronically, stored in a document management system or reproduced by any other means without prior written permission from denodo.
INDEX

PREFACE ............................................................................................................................................................................I
SCOPE ...................................................................................................................................................................................I
WHO SHOULD USE THIS DOCUMENT ............................................................................................................................I
SUMMARY OF CONTENTS ....................................................................................................................................................I

1  INTRODUCTION ..................................................................................................................................................................... 1

2  GENERAL ARCHITECTURE ....................................................................................................................................................... 2
   2.1  PHYSICAL LEVEL ............................................................................................................................................................... 3
   2.2  LOGICAL LEVEL ................................................................................................................................................................. 4
       2.2.1 Data Module: Cache ................................................................................................................................................... 5
   2.3  USER LEVEL ......................................................................................................................................................................... 6

3  INSTALLATION AND INITIAL CONFIGURATION .................................................................................................................. 7

4  EXECUTION ................................................................................................................................................................................... 8
   4.1  EXECUTING THE VIRTUAL DATAPORT GRAPHIC ADMINISTRATION TOOL ...................................................... 8
   4.2  DOCUMENTATION AND ONLINE HELP ......................................................................................................................... 10

5  SERVER ADMINISTRATION ...................................................................................................................................................... 11
   5.1  CONFIGURING THE SERVER ........................................................................................................................................ 11
       5.1.1 Port .................................................................................................................................................................................. 11
       5.1.2 Thread Pool .................................................................................................................................................................... 12
       5.1.3 Configuring the Cache ................................................................................................................................................... 13
       5.1.4 Limiting the number of concurrent queries ................................................................................................................ 15
       5.1.5 Default configuration of http Proxy ......................................................................................................................... 15
       5.1.6 Configuring the swapping to disk parameters ......................................................................................................... 16
       5.1.7 Configuring Runtime Parameters for Stored Procedures ........................................................................................... 17
       5.1.8 Configuring default internationalization .................................................................................................................... 18
   5.2  CONFIGURING LOGS .......................................................................................................................................................... 19
   5.3  EXPORTING / IMPORTING THE SERVER METADATA .................................................................................................... 19

6  VISUAL CREATION OF VIEWS ............................................................................................................................................. 21
   6.1  ACCESS TO THE GRAPHIC ADMINISTRATION TOOL .................................................................................................. 21
   6.2  CREATING AND MANAGING PROJECTS .......................................................................................................................... 21
       6.2.1 Changing elements to other projects .......................................................................................................................... 22
   6.3  IMPORTING Datasources AND CREATING BASE RELATIONS ................................................................................. 22
       6.3.1 Importing JDBC-Type Sources .................................................................................................................................. 23
       6.3.2 Importing Web-Service-Type Sources ........................................................................................................................ 25
       6.3.3 Deleting a Data Source .................................................................................................................................................... 26
       6.3.4 Creating Base Relations using a Database ................................................................................................................... 26
       6.3.5 Viewing the schema of a view ...................................................................................................................................... 28
       6.3.6 Creating Base Relations from a Web Service .............................................................................................................. 30
   6.4  VISUAL CREATION OF VIEWS AND QUERIES .............................................................................................................. 32
       6.4.1 Creating Union-Type Views ........................................................................................................................................ 32
       6.4.2 Viewing the Schema of a view .................................................................................................................................... 34
       6.4.3 Creating Join-Type Views ........................................................................................................................................... 34
FIGURES

Figure 1  DataPort Architecture ................................................................. 3
Figure 2  Authentication screen in DataPort Administration Tool ......................... 9
Figure 3  Port Configuration ........................................................................ 12
Figure 4  Cache configuration ..................................................................... 14
Figure 5  Concurrent Requests Configuration .................................................. 15
Figure 6  HTTP Proxy Configuration ............................................................. 16
Figure 7  Swapping Configuration ................................................................. 17
Figure 8  Configuration of Stored Procedures runtime parameters .................... 18
Figure 9  Creating projects .......................................................................... 22
Figure 10  Importing a JDBC Data Source ..................................................... 24
Figure 11  Configuring the JDBC connections pool .......................................... 25
Figure 12  Importing a Web-service type data source ....................................... 26
Figure 13  Schemas and tables of a JDBC source ............................................. 27
Figure 14  Accepting the schema of a base relation ........................................... 28
Figure 15  Schema of the base view phone_inc .............................................. 29
Figure 16  Schema of the base view internet_inc ............................................ 30
Figure 17  Operations and parameters of a Web Service .................................... 31
Figure 18  Schema of the base view average_monthly_sales ............................ 32
Figure 19  Constructing the union view of phone_inc and Internet_inc ................ 33
Figure 20  Schema of the unified view of incidences ......................................... 34
Figure 21  Constructing the ‘incidences’ and ‘average_monthly_sales’ join view .... 35
Figure 22  Schema of the incidences_sales join view ........................................ 36
Figure 23  Constructing the selection view pref_clients_inc_sales .................... 38
Figure 24  Editor of selection conditions ......................................................... 39
Figure 25  Creating a constant value ............................................................. 40
Figure 26  Creating a simple condition ............................................................ 41
Figure 27  RevenueSum schema .................................................................... 42
Figure 28  Compound values editor .................................................................. 43
Figure 29  Creating a value of type getSumRevenuebyTaxIds_IN0 ....................... 44
Figure 30  Condition using compound-type values .......................................... 45
Figure 31  Result of executing the view RevenueSum_Selection ........................ 46
Figure 32  Creating aggregation view inc_grouped_by_pref_clients .................... 47
Figure 33  Creating aggregation view inc_grouped_by_pref_clients (2) .................. 48
Figure 34  Editing aggregation expressions ...................................................... 49
Figure 35  Creating an aggregation expression ............................................... 50
Figure 36  Derived attribute expressions editor ............................................... 52
Figure 37  The operation getAverageMonthlyRevenue from the Web Service sales .. 54
Figure 38  Default base view created for getAverageMonthlyRevenue ................ 55
Figure 39  Flattening the array of objects returned by getAverageMonthlyRevenue .......... 56
Figure 40  Output schema of getAverageMonthlyRevenue flattened .................... 57
Figure 41  Schema of the flattened view ......................................................... 58
Figure 42  Tree view of incidences_sales .......................................................... 59
Figure 43  Executing the view incidences_sales ............................................... 60
Figure 44  Result comprised of one sole tuple with a field of the type array of registers 61
Figure 45  Viewing the content of a field of the type array of registers ................. 62
Figure 46  Incidences view execution trace ...................................................... 64
Figure 47  Join view execution trace with errors .............................................. 65
Figure 48  Confirmation of the replacement of a view ....................................... 66
Figure 49  Providing input values to interpolation variables ............................... 67
Figure 50  Importing from an ODBC datasource ............................................ 69
Figure 51  Importing from an XML datasource ................................................ 70
Figure 52  Description of an XML source ....................................................... 71
Figure 53  Importing an ITPilot wrapper ......................................................... 72
Figure 54  Importing from a new delimited file source .......................................................... 73
Figure 55  Description of a delimited-file-type source ......................................................... 74
Figure 56  Importing a new source of type Google Mini ...................................................... 75
Figure 57  Description of the Google Mini source ............................................................... 76
Figure 58  Base View Creation in Google Mini Datasources ................................................ 77
Figure 59  Creating an Aracne data source ........................................................................ 79
Figure 60  Description of the Aracne source ...................................................................... 80
Figure 61  Creating base views in Aracne sources .............................................................. 81
Figure 62  Adding an attribute with the most relevant terms of the SEARCHABLECONTENT field ........................................................................................................... 82
Figure 63  Creating an LDAP data source .......................................................................... 84
Figure 64  LDAP data source properties ............................................................................ 85
Figure 65  Importing a CUSTOM wrapper ......................................................................... 86
Figure 66  New CUSTOM data source ................................................................................ 87
Figure 67  Creating a base view from a CUSTOM data source ........................................... 88
Figure 68  New base view from a CUSTOM data source ..................................................... 89
Figure 69  Creating a base view using an SQL query ......................................................... 90
Figure 70  Editing the value of the interpolation variables ............................................... 91
Figure 71  Specifying aliases ........................................................................................... 92
Figure 72  Advanced Configuration screen of the view average_monthly_sales ............... 95
Figure 73  Editing the Execution Plan of a view ................................................................. 97
Figure 74  Search form for a bookshop .............................................................................. 99
Figure 75  Search method for a bookshop ......................................................................... 100
Figure 76  Importing a Stored Procedure ........................................................................ 103
Figure 77  Properties of a Stored Procedure ................................................................... 104
Figure 78  Execution of the Stored Procedure CalculateAvgRevenue ............................... 105
Figure 79  Result of the execution of the stored procedure CalculateAvgRevenue ............. 106
Figure 80  Publishing a Web Service ................................................................................ 107
Figure 81  Operations created for the view inc_grouped_by_pref_clients .......................... 108
Figure 82  Creating a new database .................................................................................. 114
Figure 83  Description of a database ................................................................................ 115
Figure 84  Assigning privileges on a database ................................................................. 116
Figure 85  Creating a normal-type user .......................................................................... 117
Figure 86  Creating an LDAP-type user .......................................................................... 118
Figure 87  Description of a user ...................................................................................... 119
Figure 88  Assigning privileges to a user at database level ............................................ 120
Figure 89  Assigning privileges to a user at view and/or stored procedure level .............. 121
Figure 90  Connection to DataPort from jConsole .......................................................... 123
Figure 91  Virtual DataPort MBeans .............................................................................. 124
Figure 92  Attributes tab of the RequestsManagementInfo Mbean ............................... 125
Figure 93  Details of a query in jConsole .......................................................................... 126
Figure 94  Details of a notification in jConsole ............................................................... 127
<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>Table 1</td>
<td>Conversions between DataPort data types and Web service parameter types</td>
<td>109</td>
</tr>
<tr>
<td>Table 2</td>
<td>JDBC Drivers</td>
<td>136</td>
</tr>
<tr>
<td>Table 3</td>
<td>MySQL, IBM, Microsoft and Sybase Drivers</td>
<td>136</td>
</tr>
</tbody>
</table>
PREFACE

SCOPE

Denodo Virtual Data Port provides business applications with easy access to integrated views of various heterogeneous, distributed data, and structured and semi-structured sources. By using examples, this document introduces the reader to the installation, configuration and administration of Denodo Virtual DataPort, including how to create unified views of heterogeneous and distributed data sources.

WHO SHOULD USE THIS DOCUMENT

This document is aimed at developers and administrators that require a detailed knowledge of how to install, configure and administrate Virtual DataPort, including the creation of unified data views on heterogeneous, distributed, and structured and semi-structured data sources. The detailed information required to develop applications that use Virtual Data Port is provided in the Developer Guide.

SUMMARY OF CONTENTS

More specifically, this document:

- Presents the fundamental concepts of the data integration solutions based on Virtual DataPort.

- Describes the processes required to install and configure Virtual Data Port.

- Gives a detailed description, through examples, of how the administration tools of Virtual Data Port are used to construct unified views of data from distributed and heterogeneous sources.
1 INTRODUCTION

Nowadays, in order to support its own business processes, any organization of a certain importance makes use of a multitude of information systems, developed over a period of time, that use different data sources. Generally, these data sources are developed using different technologies (relational databases, Web services, XML documents, spreadsheets, flat files, etc.), in line with very heterogeneous data models, and include both structured and non-structured data.

Development of new services for clients or optimization of company business processes require that new systems be built that, as you would expect, need the data that are stored in these inherited data repositories.

Another important tendency to consider is the expansion of the Internet, a paradigmatic example of diffusion of information and heterogeneity of formats. Its content is often a fundamental complement to corporate systems. Particularly, if we consider that clients, suppliers and competitors are already accustomed to offering Web interface to provide access to information of great interest.

Clearly, there is room for services and technologies that facilitate efficient access to and optimum use of the totality of data available, i.e. which allow extraction thereof in real time, flexible filtration and homogenization in a unified data model ready to be automatically processed by the new applications.

Virtual DataPort is a global solution for the real-time integration of heterogeneous, dispersed, and structured and semi-structure data sources. For this, it combines various features that enjoy important synergies:

Virtual DataPort integrates the data that are relevant to the company, regardless of its origin and format; it incorporates these data into its data system, in real time or with configurable preloads, and facilitates the construction of telematic services of a high strategic and functional value, both corporate and business, which cannot be conceived without using it.

This document aims to provide a technical guide to the Virtual DataPort software and its capacity to develop solutions that integrate dispersed and heterogeneous data sources with a weak structure (structured, semi-structured and unstructured sources) that are independent and lack centralized control, even when they have limited query capacity.
2 GENERAL ARCHITECTURE

In outline, Virtual DataPort enables business applications to process a series of distributed and heterogeneous data sources, including external sources as though the data were contained in a large "Virtual" Database that allows views that combine and integrate said data to be created easily.

Virtual DataPort acts as a mediator that provides a structured and unified view of the data contained in all the data sources included in the system. The system can easily deal with a wide range of structured, semi-structured and unstructured data sources such as: databases, Web sites, spreadsheets, XML documents, Web Services, flat text files, indices on unstructured information, etc.

The system considers each data source as a relation within VirtualDataPort and allows a language highly compatible with SQL to be used. Called Denodo VQL (Virtual Query Language), it is used to create views that arbitrarily combine the data of said relations using selections, projections, unions, joins, groups, etc., thus creating unified views of the source data. In this process, Virtual DataPort is even able to work with sources that have limited query capacity (for example, in most Web sources only queries that can be made through some type of HTML form are allowed). Furthermore, the Virtual DataPort query engine is capable of querying non-structured data and combining it in the required manner with structured and semi-structured data.

When the system receives a VQL query on a previously defined unified view, it can generate an execution plan for the query, which consists of a list of subqueries that are sent to the various sources involved in real time and a series of operations combining the data obtained from each source.

Furthermore, Virtual DataPort incorporates a system (called cache module) which allows the administrator to decide on the mechanism to be used for updating the source data:

- The system can access the source data in real time, thus providing totally updated data.
- Caches can be created and configured for the sources or views as required.

Virtual DataPort also allows the updating of data sources, provided that these are capable of supporting transactions.

The data integration system provided by Virtual DataPort is modeled on 3 separate levels: the user layer, the logical layer and the physical layer (wrappers). Figure 1 provides a general overview of the system architecture. Each of these levels is described below.
2.1 PHYSICAL LEVEL

The physical level aims to achieve that, when faced with higher architecture levels, the source data behave like relations with a defined structure according to the common model used by DataPort. These tasks are carried out through what are known as wrappers. A wrapper extracts data from a source, interprets the results obtained and returns them to the system in the format required by DataPort. Furthermore, where permitted by the source, a wrapper inserts, updates and/or deletes information in a source.

The mission of the physical level is to achieve that the final data sources behave like relations with a structure defined according to a common model with regard to the higher architecture levels. These tasks are carried out through the so-called wrappers. A wrapper extracts information from a source, interprets the results obtained and returns them to the schema using a specific format.

The wrappers allow Virtual DataPort to process all the external sources in a consistent manner, without concern for their specific characteristics. For this, it allows different types of wrappers to be used — one for each type of external source.

Virtual DataPort directly supports the following types of wrappers:

- Databases: These extract data from a Remote Database via JDBC or ODBC (through the JDBC/ODBC Bridge). Furthermore, where permitted by the source, a wrapper inserts, updates and/or deletes information in a source.

- Web Services: These extract data invoking operations defined by the Web services standard.
• XML: These allow data encapsulated in XML files to be extracted following a specific DTD or schema or with no associated schema or DTD. XML documents can be accessed on the local drive or via protocols such as http or FTP. Another utility of this type of wrapper is the importing of REST-type Web services.

• Flat files: Extracts data from flat files in CSV format (Comma Separated Values) or similar. CSV documents can be accessed on the local drive or via protocols such as http or FTP.

• ITPilot: Provides access to data contained in semi-structured Web sources, allowing them to be processed almost as though they were structured databases. Wrappers of this type are generated using Denodo ITPilot [ITPILOT].

• Aracne: Denodo Aracne [ARCN] allows for the crawling, filtering, indexing and search for non-structured data contained in repositories such as a Web page, relational databases, local file systems and e-mail servers. The indexes created using Denodo Aracne may be imported directly to DataPort for querying and combining with structured and semi-structured data.

• Google Mini: Google Mini [GMINI] is Google’s corporate solution for the crawling, indexing and search for data on a Web page. The indexes created using Google Mini may be imported directly to Denodo DataPort for consulting and combining with structured and semi-structured data.

• CUSTOM (also named as MY wrappers): These extract data from a source through a Java implementation provided by the Virtual DataPort administrator. This type of wrapper allows a wrapper program to be created ad hoc for a specific source. The CUSTOM wrappers also allow for data to be inserted, updated and/or deleted.

Wrappers for databases, Web services, flat files, Aracne, Google Mini and XML are automatically created by Virtual DataPort as the corresponding data sources are imported (see 6.2). The wrappers for semi-structured Web sources can be created with Denodo ITPilot [ITPILOT].

2.2 LOGICAL LEVEL

The logical level integrates and combines the relations exported by the different wrappers to comprise the system global schema. This combination is made by defining views expressed in the Denodo VQL language [VQL] of relations exported by the physical level wrappers. These relations that represent the system sources are called base relations (or base views).

Once the base relations representing the system sources have been created, the administrator can create views that intercombine them as required, thus creating the global schema views (or derived views). It is important to point out that this process can be carried out in a recursive manner in several steps: a derived view can be used as a base to create new views, thus allowing combinations of arbitrary complexity.

Once the views of the global schema have been created by combining source data, the logical level is capable of responding to queries expressed in VQL both on derived views and on base relations.

The VQL query language is SQL-based, incorporating different extensions to handle heterogeneous and distributed data. For example, VQL includes different builds to make the querying of non-structured data and its combining with structured data easier.
When the system receives a query, it checks that it can be resolved depending on the query capacity of the sources, it then draws up the possible execution plans, selects the most suitable one and executes the query returning the results obtained to the higher level.

The logical layer of Virtual DataPort also allows for data sources to be updated using INSERT/UPDATE/DELETE operations, provided that these are capable of supporting transactions.

As it can be seen in Figure 1, the following modules can be differentiated in the logical level:

- **Query Plan Generator**: Firstly, the plan generator decides if the query received can or cannot be answered in accordance with the query capacity supported by the sources. Where it is possible, it generates the possible execution plans for the query.

- **Optimizer**: Aims to select the optimum execution plan from all the options (obtained by the Query Plan Generator). For this it takes into consideration both the materialization of each of the plans activated and its complexity. The query capacity of the sources is also considered so that operations that can be executed locally can be delegated to these, thus achieving more efficient execution and less data shuffling through the network. Other aspects taken into account are the most optimal execution strategies for join operations.

- **Query Execution Engine**: Once the optimum plan has been selected, the execution engine is responsible for putting it into practice, executing the necessary subqueries on the sources and integrating the results obtained to generate the global response. In turn, the execution engine takes into consideration that information from the sources which is already preloaded in the cache module, whereby unnecessary access to data sources is avoided, thus achieving greater efficiency.

**2.2.1 Data Module: Cache**

As mentioned above, Virtual DataPort has a system (called cache module) to store local copies of the source data as required.

This cache (also called warehouse) is stored in a Relational Database accessible through JDBC. DataPort embeds an Apache Derby [DERBY] database which can be used to store the cache information. It is also possible to use the following external DBMSs to store the cache: MySQL [MYSQL] and Oracle [ORCL]. Please check the Appendix: JDBC Drivers to see the most appropriate formats for the JDBC drivers that are included / typically used with the DataPort cache system.

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 configured. As the queries are made, the tables are completed with the tuples obtained to date from the sources. Furthermore, each view maintains a description of the tuples currently contained in the cache, whereby the system manager can identify what relation queries can be solved with locally available data.

Each view in the cache has an associated expiration time, whereby the "expired" tuples are not considered in the 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 planning how often said query should be repeated using the desired time interval.

The cache can also be disabled for a specific query or for the desired relations or sources, whereby it is always possible to access the data in the sources in real time.
2.3 USER LEVEL

The user level implements the interface between the user and the Virtual DataPort manager. Thus, its responsibility is to provide the means for the user to establish a session with the manager, send queries and obtain the results corresponding to same.

The user level receives the query issued by the user in an external VQL query language and translates it into relational algebra before submitting it to the logical level.

When the logical level returns a response to the query made, this level is also responsible for providing it to the user through the connection interfaces defined to this effect (see Virtual DataPort Developer Guide [DEV]). Denodo Virtual DataPort can be accessed via JDBC, ODBC, Web Services (see section 7) and through JAVA native API (DataPort API).
3 INSTALLATION AND INITIAL CONFIGURATION

The Denodo Platform Installation Guide document [DENINST] provides all the needed information to install Virtual DataPort, including the minimum hardware and software pre-requisites. It also includes instructions for using the installation tool and to perform the initial system configuration tasks.
4 EXECUTION

To start and stop the Virtual DataPort server, there are two options:

- The Denodo Platform Control Center (see Denodo Platform Installation Guide [DENINST]) allows, among other functionality, to start and stop all servers and tools comprised in the Denodo Platform.

- Executing the script: `$DENODO_HOME/bin/vqlserver.sh startup` (vqlserver.bat startup in Windows) to start the server. To stop it, execute `$DENODO_HOME/bin/vqlserver.sh shutdown` (vqlserver.bat shutdown in Windows).

The cache system must be properly configured in the Virtual DataPort administration tool for its appropriate use (see section 5.1.3) and, if using an external DBMS to store the cache information, it must be started prior to its use. The first time Virtual DataPort is executed with cache, the relational tables required for its operation are automatically generated.

To start the Virtual DataPort administration tool, there are two options.

- To use the shortcut included in the Denodo Platform Control Center [DENINST].

- To use the script: `$DENODO_HOME/bin/vdpadmin.sh` (vdpadmin.bat or vdpadmin.exe in Windows).

4.1 EXECUTING THE VIRTUAL DATAPORT GRAPHIC ADMINISTRATION TOOL

Once the Virtual DataPort graphic administration tool has been executed, the first step is authentication in the DataPort server on which the administration tasks are to be carried out (see Figure 2). For this, the server uri should be entered together with a user login and password.

The server uri format is as follows: `//name_machine:port/database` (e.g. `//localhost:9999/admin`) where:

- `name_machine` is the name of the machine in which the server resides.
- `port` is the number of the server execution port.
- `database` (optional) is the Virtual DataPort database to which you wish to connect. When a Virtual DataPort server is installed, a default database called `admin` is created.
For more details on the concepts of database and user in Virtual DataPort, see section 7. If this is the first access to the server, the default administrator user can be used (user login “admin” with access code “admin”). We recommend that the user change this password when connecting to the server for the first time. For this, the option ‘Change password’ on the application ‘File’ menu should be used.

There are two modes for connecting to the server: database administration mode and server administration mode. The database administration mode is accessed, when the server uri specifies a database and, in addition, the specified user is not an administrator. This mode allows sources, base relations and unified views that form part of a specific database of the DataPort server to be managed.

The server administration mode is reserved for administrator users and, in that case, it is not mandatory that the supplied uri specifies a database. This mode allows tasks like those listed below to be executed:

- Create, modify or delete databases (see section 7).
- Create, modify or delete users and manage their database and view access privileges (see section 7).
- Configure the cache system (see section 5.1.3).
- Configure the Swapping policy (see section 6.6.3).
- Configure the server thread pool (see section 5.1.2).
- Configure the server execution and stop ports (see section 5.1.1).
- Configure the internationalization options (see section 5.1.8).

If the specified URI includes a database and the user is an administrator, then the administration tool will allow to manage both the server and the specified database.
4.2 DOCUMENTATION AND ONLINE HELP

Online help for the Virtual DataPort administration tool can be accessed by choosing the “Online Help” option on the “Help” menu. It is also possible to access contextual help information about the current screen by clicking on the ‘?’ icon on the upper-right section of the screen. An Internet connection is required in order to access the online help content.

The online help is organized in topics according to the table of contents shown on the left side of the Help screen. It is also possible to perform text-based searches on the help contents by using the ‘Find’ tab.

From the ‘Help’ menu, the page in Denodo’s website containing the up-to-date product documentation is also available. To access it, choose the ‘Denodo Platform Documentation’ option on the “Help” menu.
5 SERVER ADMINISTRATION

This section describes the main options available to administer the Denodo Virtual DataPort server. Firstly, details are given as to how the visual administration and creation tool for views is used to configure the various aspects of the Virtual DataPort server. How to configure the application logs is described later. Finally, how to export the Server metadata for backup and migration purposes is described.

5.1 CONFIGURING THE SERVER

To manage the databases, users and access rights of a Virtual DataPort server it is necessary to access the administration tool in server administration mode, which implies doing it with an administrator-type user. When accessing the server for the first time, the default administrator-type user can be used (user login “admin” with password “admin”).

The server configuration parameters can be classed as indicated in the following sections: port, multithreading system, cache, limiting the number of concurrent queries (query queueing), HTTP proxy, swapping parameters, stored procedures execution parameters and internationalization configuration.

5.1.1 Port

The DataPort server uses three port numbers for client communications: the server execution port, the server stoppage port and the auxiliary port. These ports can be configured by selecting the “Server Configuration – Server Port” option (see Figure 3).

NOTE: Where the connection between clients and the DataPort server is to pass through firewall software, this must be configured to allow for access to the execution port and the auxiliary port.
The port change will take effect the next time the Virtual Data Port server is relaunched.

5.1.2 Thread Pool

As mentioned earlier, wherever possible, the Virtual DataPort query execution engine operates concurrently on the data sources. Implementation of the concurrency is carried out through a multithreading subsystem that can be configured through the Virtual Data Port administration tool.

To configure the multithreading subsystem of a Virtual DataPort server access "Server Configuration - Threads Pool". The configurable parameters are as follows:

- **Max Threads**: Maximum number of threads managed in a specific moment in time by the Pool (300 by default).

- **Expiration Time**: Maximum time an unused thread remains in the Pool (600,000 milliseconds by default).

- **Sleep Time**: Indicates how often the threads not used in the Pool are checked to eliminate those that exceed the expirationTime value (300,000 milliseconds by default).

- **Timeout**: Maximum time allowed by the Pool for execution of one of the threads (600,000 milliseconds by default).
• **Checkout Time**: Maximum time a free thread is waited for. If this time is exceeded, an exception is produced (30,000 milliseconds by default).

The configuration changes will become immediately effective.

### 5.1.3 Configuring the Cache

As mentioned earlier, Virtual DataPort incorporates a system (called cache module) to store local copies of the data from sources as required in a JDBC database. To configure a server to use cache access the “Server Configuration” tab, select the “Cache” option (see Figure 4) and mark the option “Cache status on”.

Additionally, it is necessary to specify the data to create a JDBC DataSource (see section 6.3.1) for the cache. This involves specifying:

- JDBC database management system which will be used as cache. DataPort embeds an Apache Derby [DERBY] database which can be used to store the cache information by selecting the “Embedded Derby Server” option in the “DB Adapter” drop-down list. In this case, it is not required to fill any other field. It is also possible to use an external DBMS to store the cache: the Oracle and MySQL management systems are supported in the current DataPort version.
- URI to the database used as cache.
- User login and password used to access the cache Database. If an external DBMS is used to store the cache, be sure of choosing an user with read/write privileges on the specified database, including permissions to create and delete tables and indexes.
- If the ‘Test Connection’ option is marked (recommended), the system will connect to the source to obtain its manufacturer and version number. These data may be used to optimize access to the source depending on its specific characteristics. If this option is not marked, a generic configuration will be used.
- The ‘Pool Configuration’ option can also be accessed to configure the parameters of the connections pool on the cache (see section 6.3.1).
It is also possible to consult and/or configure the following cache function parameters:

- **Maintainer Period**: specifies (in seconds) the execution period of the cache maintenance task. The maintenance task is that charged with deleting entries expired in the cache. If the value provided is 0 or negative, the maintenance task will never be executed.

- **Time To Live**: specifies (in seconds) the default expiry time of cache inputs.

If the cache is stored in the embedded database, some parameters can be configured by clicking on the "Embedded Derby Server Configuration" option:

- **Enable Remote Access**: it allows the embedded database to be accessed by external JDBC-compliant clients. This can be useful for auditing and monitoring purposes.
- **Server port**: this option allows changing the port where the embedded database runs.

Once the cache has been activated and configured, the cache for individual views can now be configured for use (see section 6.6.2).

Likewise, to disable the cache simply select the “Cache off” option in the “Cache” panel of the “Server Configuration” tab.

The configuration changes will become immediately effective except when the cache manager is changed, in which case the server must be restarted.

A particular cache configuration can be specified for each DataPort server Database (see section 8.3.2).
5.1.4 Limiting the number of concurrent queries

It is possible to limit the number of requests that a DataPort server will accept concurrently. When the limit is reached, the new requests will be queued and executed according to their arrival order, whenever previous requests are finished.

Limiting the number of concurrent requests is very useful in high load environments, since it avoids performance degradation issues when there is a peak load.

To configure this behaviour, it is necessary to access the "Server Configuration" tab and cose the option "Concurrent Requests" (see Figure 5). It is then possible to configure the following parameters:

- Activate/deactivate the limitation of concurrent requests. If deactivated, the server will try to execute all requests right when they come.
- Maximum number of concurrent requests. The DataPort server will only accept this number of concurrent requests.
- Maximum number of requests on queue. Maximum number of queued requests waiting to be executed. New requests beyond this number will be discarded.

![Figure 5 Concurrent Requests Configuration](image)

5.1.5 Default configuration of http Proxy

Some VDP DataSources can use http routes to access source data. It is possible to configure the default proxy preferences for these connections by accessing the "Server Configuration" tab and choosing the "HTTP Proxy" option (see Figure 6). The configurable parameters are:

- Host. Name of the machine that acts as Proxy.
- Port. Port number in which the Proxy Server is being executed.
- Login. User identifier used to authenticate in the proxy. If the proxy does not require any kind of authentication, it must be kept blank.
- Password. User access password. If the Proxy does not require any kind of authentication, it must be kept blank.

In the http route used for each DataSource, it is possible to specify Proxy configurations different from the default one. Please see section 6.5.1 for more information.

**Figure 6** HTTP Proxy Configuration

### 5.1.6 Configuring the swapping to disk parameters

To avoid memory overflows, Virtual DataPort can swap the intermediate results generated during the run process of a view or a query to disk.

To configure the swapping parameters of a server, go to the “Server Configuration” tab and select the “Swapping” option (see Figure 7).

To enable or disable swapping on a global level, use the options “Swapping status on” (default option) or “Swapping status off”, respectively.
It is also possible to consult and/or configure the following swapping function parameters:

- Maximum size of each intermediate result (megabytes). As a view or query is being executed, DataPort will swap to disk when an intermediate result produced during the execution exceeds the maximum size specified in this parameter. As a general rule, the specified value should be no greater than one third the memory available for the JAVA virtual machine on which the DataPort server is run.

- Maximum size of the blocks written to disk (kbytes). Where DataPort swaps an intermediate result, it will write to disk using blocks of, at most, the size specified.

The general swapping configuration can also be specified for each DataPort server Database (see section 8.3.2).

If the swap is globally enabled, it is possible to enable and configure it for one view (see section 6.6.3). It is also possible to configure it dynamically for a specific query using the VQL CONTEXT clause (see the Advanced VQL Guide [VQL]).

The configuration changes will become immediately effective.

### 5.1.7 Configuring Runtime Parameters for Stored Procedures

DataPort allows executing stored procedures written in JAVA (see section 6.7 and Advanced VQL Guide [VQL]). Stored procedures can execute sentences on the DataPort server and process their results. The execution of those sentences is affected by several configuration parameters.
To configure the stored procedures runtime parameters, go to the "Server Configuration" tab and choose the "Stored Procedures" option (see Figure 8).

**Figure 8** Configuration of Stored Procedures runtime parameters

It is possible to configure the following parameters:

- **Query Timeout**: Maximum time (in milliseconds) the stored procedure will wait for the termination of a sentence. If the value 0 is specified, then the procedure will wait indefinitely until the sentence ends.

- **Chunk Size**: DataPort can return the results obtained by executing a sentence divided into blocks, so it is not needed to wait until a sentence ends to process the already obtained tuples. This parameter establishes the maximum number of results that a block can contain. If DataPort has obtained enough results to complete a block, they will be immediately returned to the stored procedure. The next results will be returned in new blocks. If this parameter is not specified (or receives the value 0), then all the results of the query will be returned in a single block.

- **Chunk Timeout**: This parameter establishes the maximum time (in milliseconds) the server will wait before returning a new block. If this time is surpassed, DataPort will return the current block even if it still does not contain the number of results specified by the ChunkSize parameter. If the ChunkTimeout parameter is not specified (or receives the value 0), then all the results of the query will be returned in a single block when the sentence execution ends.

### 5.1.8 Configuring default internationalization

The DataPort server internationalization configuration specifies aspects such as time zones, languages or currencies. For example, the default currency used in the Money-type values is specified by this property.

DataPort includes typical internationalization configurations and also allows new configurations to be created *ad hoc*. See the *Advanced VQL Guide* [VQL] for more information.
A specific internationalization configuration can also be specified for each DataPort server Database (see section 8.3.2).

At last, it is important to highlight that it is also possible to specify a different internationalization configuration for a base relation of the system (see section 6.6.1).

Lastly, the DataPort graphic administration tool also includes its own internationalization configuration that can be changed in the “View-Preferences” menu.

### 5.2 CONFIGURING LOGS

The Virtual DataPort software uses the [Jakarta-log4j][LOG4J] log system, from which it inherits its configuration. This system allows log messages to be controlled with different levels of granularity and is totally configurable using external configuration files.

Some knowledge of log4j is required to modify the log configuration file. Briefly, Log4j allows a series of categories to be defined on which the log level will be specified independently. Log4j defines different log levels that, when ordered from greater to lesser level of detail, are: DEBUG, INFO, WARN, ERROR and FATAL.

**NOTE**: In systems in production we recommend that an ERROR log level be used. Lower levels degrade the system performance and are only recommended for debugging tasks.

The Virtual DataPort Server logs are stored by default in the path DENODO_HOME/logs/vdp, where DENODO_HOME is the base installation path. The Virtual DataPort Administration tool logs are stored by default in the path DENODO_HOME/logs/vdp-admin.

The log4j.xml configuration file of the DataPort Server logs can be found in the path DENODO_HOME/conf/vdp. This file is compliant with Log4j. Among other options, it is possible to modify the path where logs are stored and the log level of each of the categories defined in the application. See Log4J [LOG4J] documentation for more information.

The log4j.xml configuration file of the Virtual DataPort Administration tool can be found in the path DENODO_HOME/conf/vdp-admin.

### 5.3 EXPORTING / IMPORTING THE SERVER METADATA

To export all server metadata to a VQL file, you can use the ‘Export…’ option in the ‘File’ menu of the graphical administration tool. It will export the metadata from every database of the Server, along with their users and their permissions. This option is available only for “administrator” users.

Executing the exported VQL file on a Virtual DataPort server, the metadata defined on the original server at the moment of the export will be recreated. This is especially useful for migration and backup purposes.

It is also possible to export the metadata from the active database using the “Export Database” option in the ‘File’ menu. In this case, it is only required to have privileges over the database to export. The metadata from a specific database can be also exported from its configuration screen (see section 8.3.2).

DataPort also provides scripts to program the running of automatic backup copies (see section 10.3).

To import the exported metadata, you need to execute the VQL file obtained during the export process. This can be done by using the VQL Shell tool from the DataPort administration tool (see section 6.4.9.1) or by using the import
script included in the utilities for importing and exporting metadata (see section 10.3). It is strongly recommended to switch to *single user* mode before importing metadata (see Advanced VQL Guide [VQL]).
6 VISUAL CREATION OF VIEWS

This section describes how to use the Virtual DataPort graphic administration tool to create unified data views graphically from various heterogeneous sources.

We will use the following example as a guide when describing the process:

**Example:** Unified incidences and client sales data

A telecommunications operator offers, amongst other things, telephony and Internet services to its clients. Data on the incidences reported in the telephony service are found in a relational database which is accessed through JDBC. Data on the incidences reported in the Internet service are found stored in another relational database also accessed through JDBC.

In our example, the director of the systems department wants to monitor the number of incidences (either telephony or Internet) notified by the clients with the greatest sales volume to establish whether or not measures should be taken to increase client satisfaction.

Data on client sales volumes are managed by another of the operator’s departments. Said department offers the remaining departments a Web Service interface to access the data in question.

In this example, we will see how Virtual DataPort can be used to construct a unified data view to meet the needs of the systems department, allowing the total number of incidences from clients with the greatest sales volumes to be obtained.

The path `{DENODO_HOME}/samples/vdp/incidences` contains SQL scripts for creating the tables used in the example in MySQL 4.1 and Oracle 9.0.1. A WSDL file is also included with the description of the Web service used, together with a `.war` file with implementation of same that has been tested with Tomcat 5.x [TOM] and Axis 1.x [AXIS].

6.1 ACCESS TO THE GRAPHIC ADMINISTRATION TOOL

The first step is to access the tool in `database administration mode` (see section 4.1). If a database and users have not yet been created (see section 7), the default database (called `vdp`) and user (`admin`) created when a DataPort server was installed can be used.

For the actions studied throughout the example, the tool must be accessed by a user that has sufficient privileges for the Virtual DataPort database being worked on (see section 7). Administrator users (like the default user `admin`) have all privileges, which means that they will always be valid.

6.2 CREATING AND MANAGING PROJECTS

The different components (datasources, base relations and views) that can be created using the DataPort administration tool are grouped together into `projects`.

On starting the tool for the first time, there will only be one project available called ‘Default’. To create a new project, select the ‘Project Management’ tab and click on the option ‘new’ (see Figure 9).
On this screen, give a name to the project and, optionally, a description. The new project will be created on clicking ‘ok’. In our example, we will create a new project that we shall call ‘VDExample’.

![Creating projects](image)

**Figure 9** Creating projects

A project can be edited by clicking on its name in the drop-down menu on the left of the screen or deleted by using the ‘drop’ option. If a project is deleted, the elements it contains are automatically moved to the ‘default’ project.

**NOTE:** these projects only exist in the client tool used to create them.

### 6.2.1 Changing elements to other projects

As can be seen in subsequent sections, on creating a new datasource or a new view, the new element created can be allocated to one of the existing projects.

Proceed as follows, if the element is later to be moved to another project:

1. Drop down the menus on the left of the screen until the required element has been located. The datasources will be accessible from the ‘Data Sources’ tab, whereas the views will be available from the ‘Query Builder’ tab.

2. Click with the right-hand button of the mouse on the element to be moved and use the ‘Move to Project’ option to select the destination project.

### 6.3 IMPORTING DATASOURCES AND CREATING BASE RELATIONS

The first step in our example is to import the sources in question from the different types and create base relations (also called base views) which represent data necessary to construct the unified views in DataPort.
To execute these tasks go to the ‘Data Sources’ tab of the administration tool. We will start by looking at how to import ‘Database’ sources, which will allow us to import databases of incidents in our example (section 6.3.1). Then the case of importing Web-Service-type sources will be dealt with (section 6.3.2), which will help us in importing data on sales in our example. Section 6.3.3 describes how to delete data sources. The differences between importing other types of data sources, as well as certain advanced aspects, will be dealt with later in section 6.5.

Once the data sources have been added, how base relations are created for same is dealt with. Section 6.3.4 deals with the creation of base relations from Database-type sources, while section 6.3.6 deals with creating base relations from Web Services. The case of other source types, as well as specific advanced aspects, will be dealt with later in section 6.5.

### 6.3.1 Importing JDBC-Type Sources

To create a new DataSource, access the “Data Sources” tab on the left of the tool. A list of created projects will appear. Click on the project to which the new source is to be added. In our example, select the previously created ‘VDPExample’ project.

The “Relational DBs” menu can then be dropped down. When said menu is opened, the list of external databases added as sources in the Virtual DataPort database will appear classified according to whether access to same is carried out through JDBC or ODBC.

To add a new JDBC database to the list click on the ‘New’ option that appears beside the JDBC database list. You then access the screen for creating a new JDBC data source (see Figure 10). The following data are then requested in this screen:

- **Name** to be given to the data source in Virtual DataPort. In our example, when the telephone service incidents database is added, we can use the name `telcenter`. When the Internet service incidents database is added, we can use the name `internet_center`.
- **Database Adapter.** In this selectable it is possible to choose one of the DBMSs DataPort includes a specific adapter for. A “Generic” adapter is also included for JDBC-accessible DBMSs without an specific adapter. Each specific adapter implements several optimizations for its target DBMS, so they should be used whenever possible. If the “Choose automatically” option is selected (recommended), DataPort will discover the suitable adapter by connecting to the DBMS. On selecting one of the adapters, the two following data fields in this screen (Path to the `.jar` file and Name of the JDBC driver class) are completed automatically.
- **Access uri** to the database. In our example, assuming that a MySQL database is used, where the incidences database is called `phone_center`, and that this is located in the machine `acme` we will write: `jdbc:mysql://acme/phone_center`. See the Appendix: JDBC Drivers for the values that must be used with the JDBC drivers that are included and/or most commonly used with DataPort.
- **User login** to access the external database.
- **User password** to access the external database.
- If the ‘Choose automatically’ option is marked (recommended), the system will connect to the DBMS to discover the suitable adapter. On marking this option the ‘Test Connection’ option is automatically selected.
Finally, if the ‘Test Connection’ option is marked, the system will connect to the source to verify it can be accessed from the DataPort server.

Figure 10 Importing a JDBC Data Source

Optionally, the option 'Driver Properties' may be used to add properties (field/value pairs) which will be added to the connection URI.

In addition, the “Connections Pool Configuration” option can also be accessed to configure several parameters of the connections pool that Virtual DataPort will use to optimize access to the external database (see Figure 11). The parameters that can be configured are:

- **Initial Size**: number of connections with which the pool is to be initialized. A number of connections are established and created in "idle" state, ready to be used.
- **Max Active**: maximum number of active connections the pool can manage at the same time (zero implies there is no limit).
- **Ping Query**: SQL query used by the pool to verify the status of the connections that are cached. The query should consume as little resources in the database as possible, and the table in question should exist.
- **Test Connection**: if this property is checked and there is a specified ping query, each connection retrieved from the connection pool will be validated by executing the ping query.

**NOTE:** Using Virtual DataPort version 3.5, the remaining parameters that were configurable in previous versions are no longer so.

For our example, we will leave all the parameters with their default value for both databases.
Clicking on the ‘OK’ button in the data source creation screen causes this to appear in the list shown in the menu on the left side of the screen.

### 6.3.2 Importing Web-Service-Type Sources

Continuing with our example, the next step is to add the data source which allows us to access the sales data. As mentioned earlier, this source is of the Web Service type.

To create a new data source representing a Web Service, access the “Data Sources” tab and, after selecting the project to which the new source is to be added (VDPExample in this case), click on the ‘New’ option that appears beside the list of current Web Services. You then access the new Web-Service-type data source creation screen (see Figure 12). The following data are requested in this screen:

- Name to be given to the data source in Virtual DataPort. In our example, when the Web service telephone service incidents are added, we can use the name `sales`.
- Path to the WSDL file containing the Web service specification. In our example, the path could be `DENODO_HOME/samples/vdp/incidences/sales.wsdl`. An http address can also be indicated.
- If the access to the web service requires authentication, the user identifier and password must be provided, as well as the authentication protocol that must be used. DataPort supports two kinds of authentication modes for Web Services:
  - HTTP. Basic mode http authentication will be used [HTTP-AUTH].
  - WSS. Web Services Security [WSS] is a Standard for the implementation of security features in applications using Web Services. WS-Security offers a mechanism which associates content with security information. Currently, Denodo supports the authentication profile called “Username Token” [WSS-UT].
• If the access to the Web Services is realized through a Proxy, the host name and port where the Proxy is executed must be provided. If it is an authenticated proxy, the user identifier and a valid password must also be specified. It is also possible to use the default http proxy configuration (see section 5.1.5) by clicking the option “Default”.

![Virtual DataPort Administration Tool](image)

Figure 12 Importing a Web-service type data source

### 6.3.3 Deleting a Data Source

Once a data source has been added, it is possible to delete it by clicking on the ‘drop’ button appearing besides the data source name in the list under the ’DataSources’ menu.

If there exist other elements in the DataPort catalog (e.g. views) which depend on the data source, the user will receive a warning message informing her that the operation will be executed on cascade. If the user decides to proceed with the operation, the elements having dependencies with the data source will also be deleted. If the user does not have the required privileges over all the involved elements, then the operation will fail.

### 6.3.4 Creating Base Relations using a Database

Once a data source has been added to the system, it can be used to create base relations. For this, you should click on the data source in the list on the left side of the screen to access its contents.

In the case of database-type sources, when this action is carried out a tree will be shown with the various schemas making up the external database, allowing the tables comprising it to be viewed as well as the fields that constitute them (see Figure 13).
A 'Create base view' button appears beside each table of the external database. If this button is pressed, the system automatically generates the schema of a base relation from the table selected and shows it to the user (see Figure 14). Where required, the name of the base relation and the name and type of the attributes comprising it may be changed at this stage. The project to which the new view is to be allocated can be specified using the selectable 'Target Project'. In our example, select the 'VDPExample' project. The base view is created just by clicking 'ok'. If the user wishes to create a base view and then import another one from the same DataSource, he/she can press the 'OK and import new view' button, which will create the view and will go back to the screen where the schema and database table tree is shown.

Once the schema has been accepted, the new base relation can be seen in the left part of the screen hanging from the source (database, in this case) to which the table used belongs.

In our example, we will create a base relation for the telephony database table **phone_inc** and another for the Internet database table **internet_inc**.

**phone_inc** provides access to data on the incidences registered in the telephony service. It includes fields that represent the client Tax ID (**taxId**) that reported the incidence, an incidence identifier (**pinc_id**), a description of same (**description**), the time at which it occurred (**time**), an incidence type (**inc_type**) and an additional specific field (**specific_field3**).

The attributes of **internet_inc** include the Tax ID (**taxId**) of the client that reported the incidence, an incidence identifier (**inc_id**), a description of same (**summary**), the moment in time at which it occurred (**time**) and another two additional specific fields (**specific_field1** and **specific_field2**).

It is also possible to create a base relation from a SQL query to the database ('Create Base View From Query' button). The use of this option is explained in section 6.5.10.
6.3.5 Viewing the schema of a view

Clicking on a view we have created we can access the page that shows its schema in Virtual DataPort (see Figure 15 and Figure 16 with the schemas of the base views generated from phone_inc and internet_inc).

Once the view has been created, a projection view of same should be used to modify its schema to eliminate attributes or change their name (see section 6.4.6). Pressing the ‘Edit’ button provides direct access to the projection view creation screen for the view. Another alternative is to replace the view for another with the same name (see section 6.4.11).

Other possible actions from this screen are:

- Execute the view (‘Execute’ button). Obtains the view tuples. The system will execute a ‘select * from viewname’ query by default, where viewname is the name of the view. However, the tool also offers the option of accessing the conditions editor before execution to create a selection condition for the view (see section 6.4.4.1).
- Delete the view (‘Delete’ button). If there are other derived views using the view to be deleted in their definition, the tool will warn the administrator that deleting the view will cause the deletion on cascade of the derived views.
- See the VQL code generated to create the view (‘View VQL’ button).
- Hide the view (‘Hide View’ button). If the button is pressed, the view will not be shown in the list on the left of the screen. To view it once again the menu ‘View Hidden Views’ can be used, selecting the ‘hidden’ view you want to reappear.
From this screen it is also possible to access the advanced configuration aspects of the view ('Advanced' button), although this option will not be explained until section 6.6.

Figure 15 Schema of the base view phone_inc
6.3.6 Creating Base Relations from a Web Service

Once a Web Service data source has been added to the system, it can be used to create base relations. For this, you should click on the data source on the list on the left side of the screen to access its contents.

In the case of Web-Service-type sources, when this action is carried out, the different Web Services contained in the WSDL file and the various ports that make up each of these are shown. The list of operations included is shown for each port. The input parameters, output parameters and input/output parameters are shown in the details of each operation (see Figure 17).

In our example, the `sales` Web service comprises one port that contains several operations. The operation we are interested in receives the name `getAverageMonthlyRevenueBytaxId`. It has an input parameter called `in0` that specifies the client Tax ID (`taxId`) and returns the average monthly volume of sales.

Beside each Web Service operation a ‘Create base view’ button appears. If this button is pressed, the system automatically generates the schema of a base relation from the selected operation, and this is shown to the user. The name of the base relation and the name and type of the constituent attributes may be modified at this stage. The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

In our example, we will change the name of the base relation for that of `average_monthly_sales`, that of field ‘in0’ for ‘taxId’ and that of field ‘GetAverageMonthlyRevenueBytaxIdReturn’ for ‘revenue’.

![Figure 16 Schema of the base view internet_inc](image-url)
The base view is created just by clicking ‘ok’. If the user wishes to create a base view and then import another one from the same DataSource, he/she can press the ‘OK and import new view’ button, which will create the view and will go back to the screen where the Web Service operation lists is shown. Once the schema has been accepted, the new base relation can be seen on the left side of the screen hanging from the Web-Service-type source to which the operation used belongs.

Clicking on a base relation we have created we can access the page that shows its schema in Virtual DataPort (see Figure 18). The available options in this screen were described in section 6.3.5.

Although this is not the case in the operation used in our example, the results returned by an operation may belong to the compound data type (e.g. arrays of elements). In this case, these results will be represented in the base relation in DataPort array and register compound-type fields. At times, it can be a good idea to “flatten” these fields to achieve a more appropriate granularity in the tuples of the base relation. This process is described in section 6.4.7.

Another similar situation occurs when an operation imported from a Web Service requires compound-type parameters (e.g. an array of elements) at input. In this case, the operation input parameters will also be represented in the base relation in DataPort array and register compound types. Furthermore, the ROW constructors (to create register-type elements) must be used to build query conditions on this base relation and ‘( )’ to create type elements. Section 6.4.4.1.4 shows an example.
Once the base relations have been created with the data we require from the sources, the ‘Query Builder’ tab can be accessed to visually create derived views that combine and integrate data as required.

Views have the following characteristics:

- A view is another relation in the Virtual DataPort catalog. It may then appear in the FROM clause of any VQL query and may be used as a base for constructing new views or queries.
- The cache may be used on a view, and user privileges can be defined (see section 7).
- When creating a view the query execution plans are partly precalculated. This leads to a more efficient execution of view queries.

The following sections describe in detail the process involved in visually creating views using our example to illustrate the process.

### 6.4.1 Creating Union-Type Views

A union-type view allows the tuples from various input relations to be contained in a single view.

To carry out the union operation in standard relational algebra all the relations must have the same schema, i.e. the same attributes and that they be of the same type. However, in Virtual DataPort an extended union is used, whereby if any of the input relations has an attribute that is not present in the others, this is added to the resulting view.
To define a union view open the ‘Toolbox - Relational Operations’ menu and select the ‘Union’ option. To select the relations on which the union operation is to be executed, drag&drop from the list of views and base relations that appear on the left side of the screen. In our example, we will drag&drop from the base views phone_inc and internet_inc (see Figure 19).

As input views are added to the union view, the tool automatically generates the schema of the resulting union view. The schemas of the input views are shown on the left of the work area, while the output schema is displayed on the right.

![Figure 19 Constructing the union view of phone_inc and Internet_inc](image_url)

Below the input views a ‘List of associations’ is also shown amongst its fields. The fields united by an association are considered a sole attribute in the output schema. In order to establish an association between two fields they must be of the same type. By default, the system will create associations between the attributes of the same name and same type (in our example this occurs with the fields time and taxId).

Additional associations may also be specified between attributes which receive different names in the different input views. In our example we will do this with the attributes description of phone_inc and summary of internet_inc.

For this, simply drag&drop from one attribute to another and one sole attribute will appear in the output schema representing both. The name assigned to the attribute in the output schema will be that of the first attribute marked in the drag&drop operation.

A previously created association may also be deleted using the “[x]” button beside the association.

On the output view schema the names of the attributes can be changed, the name of the result view can be changed and the attributes not required in the view can also be eliminated. To accomplish it, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Field’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes. In our example we will eliminate the fields pinc_id, iinc_id, specific_field1, specific_field2 and...
specific_field3, as these are not necessary to our requirements. We will select incidences as the name for the resulting view.

New derived attributes can also be added to the resulting union view using the ‘New Field’ button, which provides access to the derived attribute expressions editor (see section 6.4.6.1).

Once the output view has the desired form, simply press the ‘ok’ button to create it. Clicking on it in the list on the left of the screen provides access to its schema.

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

### 6.4.2 Viewing the Schema of a view

To access the schema of a view click on the view (see Figure 20). In addition to the options available for base views (see section 6.3.5), using the ‘Tree View’ button on this screen accesses a tree representation of the view that shows how this is defined in accordance with base relations and relational algebra operators.

![Figure 20 Schema of the unified view of incidences](image)

### 6.4.3 Creating Join-Type Views

A join-type view allows the relational algebra operation with the same name to be executed on a series of input views.

To define a join view open the ‘Toolbox - Relational Operations’ menu and select the ‘Join’ option. To select the relations on which the join operation is to be executed drag&drop from the list of views and base relations that...
appear on the left side of the screen. In our example, drag & drop from the incidences view and from the base view average_monthly_sales (see Figure 21).

As input views are added, the schema of the resulting join view is generated automatically. The schemas of the input views are shown on the left of the work space, while the output schema is shown on the right.

To add a join condition the operator involved in the selectionable that appears beside the text “Operator” must be specified and then dragged & dropped to link the join attributes in the input views. The name assigned to the attribute in the output schema will be that of the first attribute marked in the drag & drop operation.

In our example we will select the operator ‘=’ and drag & drop from the attribute taxId of the incidences view to the attribute taxId of the average_monthly_sales view, thus defining the join condition incidences.taxId=average_monthly_sales.taxId.

The join operation type required can also be specified: inner, left outer, right outer or full outer (see ‘FROM Clause’ in the Advanced VQL Guide [VQL]). A conventional ‘inner join’ is used in our example.

The preferences on the join run strategy can also be established. This election may have significant implications on the optimization of queries (see ‘Query Optimization’ in the Advanced VQL Guide [VQL]). This involves two parameters:

- Execution method (merge, hash, nested or nested parallel). Where the nested_parallel method is chosen, the maximum number of parallel subqueries to be run on the second source can also be chosen (see ‘Query Optimization’ in the Advanced VQL Guide [VQL] for further information on the nested parallel run method).
- Order in which the input relations must be considered. The view at the top will be considered as a first relation, if the Ordered option is chosen, and as second, if Reverse Order is chosen.)
In our example, no preference will be established for the run strategy, leaving the VDP server to select the one it considers best.

The names of the attributes can be changed on the output view schema, the name of the resulting view can be changed (by default it will be vista1_j_vista2), and the attributes that are not required in the view can be eliminated (this assumes an implicit projection-type view. See section 6.4.6). To delete attributes, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Field’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes. In our example, we will give the view the name incidences_sales and we will eliminate one of the taxId attributes. New derived attributes can also be added to the resulting view using the ‘New Field’ button that provides access to the derived attribute expressions editor (see section 6.4.6.1).

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

Once the output view has the desired form, simply press the ‘ok’ button to create it. Clicking on the list on the left of the screen provides access to its schema (see Figure 22).

**Figure 22** Schema of the incidences_sales join view

6.4.3.1 Join conditions with similarity operators

DataPort supports join conditions that use similarity operators. The similarity operator assessment returns a value between 0 and 1 that estimates the similarity between the operands using a certain similarity algorithm. As well as the operands to compare, the operator receives the similarity algorithm to use and a minimum similarity threshold as parameters. Where the similarity between operands reaches or exceeds the threshold, the condition is assessed as true. Where this is not the case, it is assessed as false.
DataPort includes the operator ~ (ALT-126) to assess the similarity between character string-type operands. For this, DataPort includes algorithms based on the edition distance between the texts (ScaledLevenshtein, JaroWinkler, Jaro, Level2Jaro, MongeElkan and Level2MongeElkan algorithms), on the appearance of terms common to both texts (TFIDF, Jaccard and UnsmoothedJS algorithms) or combinations of both (JaroWinklerTFIDF). If no algorithm is specified, DataPort chooses the one to apply.

If the ~ operator is selected as the join condition operator, the similarity algorithm to be used (where the default option is chosen, the algorithm will be chosen by DataPort) and the minimum similarity threshold to be reached for the operands to be considered as paired must be specified.

**Example**: Consider a variation of the example in which customers are identified by their name instead of their taxid. Unfortunately, the name of the customers in the incidence databases does not exactly match their name in the sales Web Service and, therefore, a join operation with the equality operator will not give good results. In this case, a join can be used with a similarity operator to solve this problem.

The incidences example included with distribution also allows for this situation to be reproduced. Two additional relational tables are included known as internet_inc_cname and phone_inc_cname. These tables are similar to the internet_inc and phone_inc tables used in the example, although they include a customer_name attribute to indicate the name of the customer instead of the taxid attribute. The sales Web Service also includes an operation known as GetAverageMonthlyRevenueCName that returns a compound data element containing the name and the monthly turnover of all the customers. To solve the problem posed by this example, follow the steps below:

1. Create base views for the internet_inc_cname and phone_inc_cname tables (see section 6.3.3).
2. Create a base view for the GetAverageMonthlyRevenueCName operation (see section 6.3.6).
3. Join the base views created for internet_inc_cname and phone_inc_cname (see section 6.4.1).
4. “Flatten” the compound element returned by the view created for the GetAverageMonthlyRevenueCName operation. See section 6.4.7 for a detailed explanation of the compound data element “flattening” process.
5. Make a join view using the customer_name field between the view obtained as a result of step 3 and the view obtained as a result of step 4. Specify operator ~ as the join operator and set the similarity threshold to 0.7.
6. Run the view obtained in step 5. See how the join is made correctly, despite the fact that the customer names in the input views were not exactly the same.

### 6.4.4 Creating Selection-Type Views

A Selection-type view allows a condition to be executed on an input view.

To define a selection view, open the ‘Toolbox - Relational Operations’ menu and select the ‘Selection’ option. To select the view on which to execute the selection operation drag&drop from the list of views and base relations that appear on the left side of the screen. In our example, we will drag&drop from the incidences_sales view (see Figure 23).

To add a selection condition press the ‘Set Condition’ button, accessing in this way the conditions editor (see Figure 24), which allows us to create the desired condition (see section 6.4.4.1 for more information). Once the condition has been created, simply press the ‘OK’ button, and the condition will be added to the selection view. In our example, we will enter the condition revenue>6000 to retain only data of those clients for whom the average monthly volume of sales exceeds 6,000 euros.
The names of the attributes can be changed on the output view schema, the name of the resulting view can be changed (in our example we will select `pref_clients_inc_sales`), and the attributes that are not required in the view can be eliminated (this assumes an implicit projection-type view). To delete attributes, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Fields’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes. New derived attributes can also be added to the resulting union view using the ‘New Field’ button which provides access to the derived attribute expressions editor (see section 6.4.6.1).

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

Once the schema of the output view has the desired form, simply press the ‘ok’ button for the new view to be created and appear in the list on the left part of the screen.

6.4.4.1 Using the Conditions Editor

The conditions editor (see Figure 24) allows selection conditions to be created. The condition can be written directly in VQL format in the ‘Selection Condition’ box or can be created completely graphically. This last process is described below.

On the left side of the screen we will find menus for creating various values that can appear as operands in the conditions:

- Constants. This menu allows constants of the various data types supported by Virtual DataPort to be created.
Functions. This menu allows an invocation to one of the functions permitted by Virtual DataPort to be created. The functions can receive attributes or the result of evaluating other functions as constant parameters. They return one result. The list of available functions and use of each of them can be seen in the VQL Advanced Guide [VQL].

Attributes. This corresponds to the list of attributes of the view to which the selection is applied. The attributes can act as function parameters and/or appear directly as operands in the conditions.

On the right of the screen we will find menus to select the various operators that can appear in the conditions:

- Operators that can participate in the simple conditions. These are grouped according to the data type to which they can be applied. NOTE: the blob and xml data types are not shown in the operators right menu because they do not have any associated operator.
- Logical operators (AND, OR, NOT). These are used to combine the different simple conditions in a Boolean expression.

The center boxes of the screen allow three types of elements to be constructed from top to bottom: values that appear in the conditions, simple conditions and compound Boolean conditions. The box on the left of each group is a workspace for creating new elements, while the box on the right shows the elements already created. The following subsections describe in more detail how each of these types of elements is created.

Finally, the “Selection condition” box contains the selection condition eventually created.

6.4.4.1 Creating values for the conditions

To create a new constant value the following actions are required:
1. Select the data type of the constant in the ‘Constants’ drop-down menu on the left side of the screen and click on it or drag&drop to the work space, where values are created (box on top left).

2. The type selected will appear in the workspace together with a text area to fill in the value of the constant.
   a. Where the constant belongs to a simple data type, the value required can be written directly in the text area.
   b. Where it belongs to a compound type (array or register), a new panel will be opened on clicking the ‘□’ button alongside the text area to edit the required constant values (see section 6.4.4.1.4).
   c. If it belongs to the xml type, a panel will be opened on clicking the ‘□’ button that allows for the value required to be written or loaded from a file.

3. On clicking the ‘>’ button, the new constant will appear in the list of values created (upper right-hand box).

To create a new function-type value the following actions are required:

1. Select the required function in the ‘Functions’ drop-down menu on the left side of the screen and click on it or drag&drop to the work space for creating values (box on top left).

2. The function selected will appear in the work space together with an area to fill in the value of each parameter of the function. The values of the parameters should be values present in the list of created values (box on top right). To assign a value already created as a parameter of a function drag&drop the value created to the parameter area. Press the ‘>’ button that appears beside the function, and this will appear in the list of values created (box on top right).

In our example, we will create a new constant value of the type Float. For this we will drag the type Float to the work space and fill in the value of same, in this case it will be 1000 (see Figure 25).

![Figure 25 Creating a constant value](image-url)
To create a new simple condition the following actions are required:

1. Select the required simple condition operator in the drop-down menus on the right side of the screen and click on it or drag&drop same to the work space, where the simple conditions are created (left center box).
2. The operator selected will appear in the work space with an area to fill in its operands. The operands can be either attributes of the input view (present in the “Fields” drop-down menu of the left side of the screen) or values already created (the list of which is displayed in the box on the top right). To assign an attribute or a value already created as an operand of the condition drag&drop the element to the parameter area. Pressing the ‘>’ button that appears beside the simple condition will cause this to appear in the list of simple conditions already created (centre right box).

In our example, we will select the operator ‘>=’ associated with the type Float and drag&drop same to the work space. We will use the attribute revenue as left operand. We will use the constant value Float(1000) created in the previous section as right operand. See Figure 26.

![Figure 26 Creating a simple condition](image)

6.4.4.1.3 Creating Boolean conditions

To create a new Boolean condition the following actions are required:

1. Select the required Boolean operator (AND, OR or NOT) in the drop-down menus on the right side of the screen and click on it or drag&drop same to the work space, where the Boolean conditions are created (left lower box).
2. The operator selected will appear in the work space together with an area to fill its operands. The operands can be simple conditions already created (the list of which is shown in the right center box) and other Boolean conditions created beforehand. To assign a condition already created as an operand of the new Boolean condition drag&drop the condition to the operand area. Press the ‘>’ button that appears
beside the Boolean condition, this then appears in the list of Boolean conditions already created (right lower box).

Finally, drag&drop the condition to be added to the selection to the “Selection Condition” box. Press ‘ok’ to return to the selection view creation page, where the condition will appear already created.

In our example, we will not create compound Boolean conditions, whereby we will directly drag&drop from the simple condition created in the previous section to the “Selection Condition” box.

6.4.4.1.4 Creating conditions that use compound-type constants

Denodo Virtual DataPort supports the modeling of data types with tree structure using register and array types (see Advanced VQL Guide [VQL]).

It is possible to create constants of these compound types and to use them in the selection conditions. An explanation of how to do so will be given in this section through an example.

![Virtual DataPort Administration Tool](image)

Figure 27 RevenueSum schema

In the example, a base relation called RevenueSum will be used, the schema of which is shown in Figure 27. RevenueSum can be created as follows:

1. Using the datasource to access the Web Service in the sales example (see section 6.3.2), create a base relation based on the getSumRevenueByTaxIds operation. This operation is given an array of character strings (each one will be the taxId of one client) as input and returns the sum of the revenue of all specified clients (float-type value).
2. The new base relation will be called RevenueSum. The name of its attributes shall also be changed for clients and totalrevenue, respectively. The clients attribute belongs to the ‘register array’ data type and represents the operation input parameter. More specifically, DataPort automatically generates an array type called getsumrevenuebyids_in0. Each element in this array type will be of a register type, which is also automatically generated by DataPort and called getsumrevenuebyids_in0_string. The elements in this last type have a single text-type field for which the name will be changed to ‘taxId’.

Figure 28 Compound values editor

In this example, a selection view will be created using a condition on RevenueSum to obtain the total obtained by adding together the revenues of the clients with taxIds ‘B78596011’ and ‘B78596012’. The steps to follow are as given below:

1. Select the option ‘Query Builder – Toolbox – Relational Operations – Selection’ and drag RevenueSum to the work area to create a new selection view. Click the ‘Set Condition’ option to access the selection conditions editor.

2. The compound data type getSumRevenuebyTaxIds_IN0 will now appear in the ‘Constants’ drop-down on the left of the screen, along with the single data types.
3. Drag & drop the `getSumRevenuebyTaxIds_IN0` type to the value creation area and click the '□' button.

4. A new pop-up will appear as shown in Figure 28 to create a new `getSumRevenuebyTaxIds_IN0` type constant. Given that this is an array type, it is possible to create several elements for it. To do so, click the '+' button alongside 'Value'. In our example, the array must have two elements and, therefore, will be clicked twice.

5. It is now possible to edit the values for each of the array elements that will always be of a register type. To do so, simply click on them and they will drop down as shown in Figure 29. There will be a text area in which the required value can be written for each register-type field. There is a single field called `taxId` in the type of our example. The value 'B78596011' will be entered in the first register and 'B78596012' in the second.

![Edit compound value of type getsumrevenuebytaxids_in0](image)

**Figure 29 Creating a value of type getSumRevenuebyTaxIds_IN0**

6. By clicking 'ok', the pop-up will close and the new compound value will appear in the creation area. By clicking the '>' button, the new constant will be created (it is represented in VQL by `{ROW('B78596011'), ROW('B78596012')}`). See the *Advanced VQL Guide* [VQL] for more information on `ROW` and `{ }` constructors.
The new constant can now be used to create the query condition required on the `clients` field. The creation process for this condition is the same as with single-type constants (see section 6.4.4.1.2). Figure 30 shows the condition created in the “Selection Condition” box (represented in VQL by: `clients={ROW(B78596011),ROW(B78596012)}`). Figure 31 shows the result of running the resulting selection view (see section 6.4.9 to obtain information about how to execute views and navigate through the obtained results when these are of compound type).

Figure 30 Condition using compound-type values
6.4.5 Creating Aggregation-type views

An Aggregation-type view allows a GroupBy-type operation to be executed on an input view.

To define an aggregation view open the ‘Toolbox - Relational Operations’ menu and select the ‘Aggregation’ option. To select the view on which the aggregation operation is to be executed drag&drop from the list of views and base relations that appear on the left side of the screen. In our example, we will drag&drop from the view `pref_cli_inc_sales`.

The schema of the input view is displayed on the left of the work area, while the current output schema is displayed on the right. At the beginning, said output schema will be empty.

It is important to remember that an aggregation-type view can have two types of attributes in its output:

- GroupBy Attributes. Attributes that execute the group-by operation.
- Attributes resulting from the application of an aggregation expression. Aggregation expressions involve the use of aggregation functions such as COUNT, MIN, AVG, etc. (see the VQL Advanced Guide to see the complete list and description of the aggregation functions).

To specify the groupBy attributes simply drag&drop over the ‘Group By Fields’ that appear in the central lower part of the screen.

An attribute of the input view can also be dragged directly to the output view. In this case, the attribute is considered a group-by attribute and will also appear in the output view schema (see Figure 32). In our example, the group-by fields will be `taxId` and `revenue`. 

![Figure 31 Result of executing the view RevenueSum_Selection](image-url)
To add attributes resulting from an aggregation expression we will use the “New Aggregation Expression” button that will give us access to the aggregation expressions editor. Once the aggregation expression is created, an attribute will be added to the schema of the output view, with the name and the expression created using the aggregation expressions editor.

In our example, we will add an attribute resulting from the aggregation expression \( \text{count}(*) \) to which we will assign the name \text{num_incidences} (see Figure 33). Use of the aggregation expressions editor is described in section 6.4.5.1.
The names of the attributes can be changed on the output view schema, the name of the resulting view can be changed (in our example we will select `inc_grouped_by_pref_clients`), and the attributes required can be eliminated. To delete attributes, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Fields’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes. New derived attributes can also be added to the resulting union view using the ‘New Field’ button which provides access to the derived attribute expressions editor (see section 6.4.6.1).

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

Once the output view has the desired form, simply press the ‘ok’ button to create it. Clicking on it on the list on the left part of the screen will provide access to its schema.

6.4.5.1 Using the Aggregation Expressions Editor

The aggregation expressions editor (see Figure 34) allows aggregation expressions to be created. The expression can be written directly in VQL in the ‘Field Expression’ box and the name of the new attribute in the ‘Field name’ box. The expression can also be created completely graphically. This last process is described below:
On the left of the screen we will find menus to create various values that can appear as operands in the expressions:

- Constants. This menu allows constants of the various data types supported by Virtual DataPort to be created.
- Aggregation functions. This menu allows an invocation to one of the aggregation functions permitted by Virtual DataPort to be created. The functions can receive attributes or the result of evaluating other functions as constant parameters. They return one result. The list of available functions and use of each of them can be seen in the VQL Advanced Guide [VQL].
- Attributes. This corresponds to the list of attributes of the view to which the aggregation is applied. Attributes can act as function parameters. The value `*` also appears in the list, which serves to construct aggregation expressions that are applied to all the tuple fields (e.g. `count(*)`).

The boxes on the top of the screen allow expressions to be constructed. The box on the left is a work space for creating new expressions, while the box on the right displays the expressions already created.

Finally, the “Aggregation Expressions” box contains the expression eventually created. This box can also include the name which will be associated in the output view with the attribute associated with the expression.

To create a new constant expression the following actions are required:

1. Select the data type of the constant in the ‘Constants’ drop-down menu on the left of the screen and click on it or drag&drop to the work space, where expressions are created (left box).
2. The type selected will appear in the workspace together with a text area to fill in the value of the constant.
   a. Where the constant belongs to a simple data type, the value required can be written directly in the text area.
b. Where it is of the compound type (array or register), a new panel will be opened on clicking the ‘□’ button alongside the text area to edit the required constant values (see section 6.4.4.1.4).

c. If it belongs to the xml data type, a panel will be opened on clicking the ‘□’ button that allows for the value required to be written or loaded from a file.

3. On clicking the ‘>' button, the new constant will appear in the list of values created (upper right-hand box).

To create a new function-type expression the following actions are required:

1. Select the required function on the ‘Functions’ drop-down menu on the left of the screen and click on it or drag&drop to the work space, where expressions are created (left box).

2. The selected function will appear in the work space together with an area to fill in the value of the parameters of the function. The values of the parameters should be either expressions present in the list of created values (right box) or attributes. To assign an expression already created as a parameter of a function drag&drop the expression created to the parameter area. Pressing the ‘>' button that appears beside the function will cause this to appear in the list of expressions created (right box).

In our example, we will create a new function-type expression with the function `count` applied to the special attribute * (see Figure 35). Finally, we will drag the expression created to the ‘Aggregation Expression’ box, and we will assign the new attribute the name `num_incidences`.

![Aggregation Expression Editor](image)

**Figure 35** Creating an aggregation expression

### 6.4.6 Creating Projection-Type Views

A Projection-type view allows attributes to be eliminated or added to an input view.
To define a projection view open the ‘Toolbox - Relational Operations’ menu and select the ‘Projection’ option. To select the view on which the selection operation is to be executed drag&drop from the list of views and base relations that appear on the left of the screen.

The names of the attributes can be changed on the output view schema, the name of the resulting view can be changed, and the attributes that are not required in the view can be eliminated. To delete attributes, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Fields’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes. New derived attributes can also be added to the resulting projection view using the ‘New Field’ button, which provides access to the derived attribute expressions editor (see section 6.4.6.1).

It is important to highlight that it is also possible to project an atomic attribute that forms part of a compound attribute (attributes of the type `Register`). This can be done either through the derived attribute expressions editor or, more quickly and easily, by clicking on the right mouse button over the required atomic field and selecting the option ‘Project subfield’.

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’. In our example, select the ‘VDPExample’ project.

Once the output view has the desired form, simply press the ‘ok’ button to create it. Clicking on the view in the list on the left of the screen provides access to its schema.

6.4.6.1 Using the Derived Attribute Expressions Editor

The derived attribute expressions editor is shown in Figure 36. The desired expression can be written directly in VQL in the ‘Field Expression’ box and the name of the new attribute in the ‘Field name’ box. The expression can also be created completely graphically. Now we describe this last process.
On the left of the screen we will find menus to create various values that can appear as operands in the expressions:

- Constants. This menu allows constants of the various data types supported by Virtual DataPort to be created.
- Derived attribute functions. This menu allows an invocation to one of the derived attribute functions permitted by Virtual DataPort to be created. The functions can receive attributes or the result of evaluating other functions as constant parameters. They return one result. The list of available functions and use of each of them can be seen in the VQL Advanced Guide [VQL].
- Attributes. This corresponds to the list of attributes of the view to which the projection is applied. The attributes can act as function parameters.

The center boxes of the screen allow expressions to be constructed. The box on the left is a work space for creating new expressions, while the box on the right displays the expressions already created.

Finally, the “Field expression” box contains the expression eventually created. This box can include the name that will be associated with the attribute associated with the expression in the output view.

To create a new constant expression the following actions are required:

1. Select the data type from the constant in the ‘Constants’ drop-down menu on the left side of the screen and click on it or drag&drop to the work space, where expressions are created (left box).
2. The type selected will appear in the workspace together with a text area to fill in the value of the constant.
   a. Where the constant belongs to a simple data type, the value required can be written directly in the text area.
b. Where it is of the compound type (array or register), a new panel will be opened on clicking the ‘□’ button alongside the text area to edit the required constant values (see section 6.4.4.1.4).

c. If it is of the xml type, a panel will be opened on clicking the ‘□’ button that allows for the value required to be written or loaded from a file.

3. On clicking the ‘>’ button, the new constant will appear in the list of values created (upper right-hand box).

To create a new function-type expression the following actions are required:

1. Select the required function in the ‘Functions’ drop-down menu on the left of the screen and click on it or drag&drop to the work space for creating expressions (left box).

2. The function selected will appear in the work space together with an area to fill in the value of the function parameters. The values of the parameters should be expressions present in the list of created values (right box) or attributes. To assign an expression already created as a parameter of a function drag&drop the expression created to the parameter area. Press the ‘>’ button that appears beside the function, and this will appear in the list of expressions created (right box).

6.4.7 Flattening of register-array-type fields

Denodo Virtual DataPort supports the modeling of data types with tree structure using the types register and array (see Advanced VQL Guide [VQL]).

In DataPort, an element of the type array can be considered a sub-view; in this way, an array type always has an associated register type that acts like the schema of the sub-view it is modeling.

Sometimes you may wish to “flatten” a compound field that contains an array of registers. This is particularly frequent when processing XML- and Web-services-type sources. This section describes how this is done.

It is important to note that if you wish to directly “flatten” a register (and not an array of registers), this should be done using the projection view (see section 6.4.6) in place of the mechanism described here.

Imagine the next modification in the example we have been using until now. We will now use the operation getAverageMonthlySales of the Web service sales. This operation is not assigned any parameter and, furthermore, the sales data from all the clients will be returned through an array of objects, where each object has two properties: taxId and revenue (see Figure 37).
Figure 37 The operation `getAverageMonthlyRevenue` from the Web Service `sales`

The default base relation created on this new operation will have one sole attribute of the type array of registers and one sole tuple in which all the data returned by the Web service are found (see Figure 38). As regards combining data with other sources (for example, to combine with tables of incidences in our example), a view that has two attributes (`taxId` and `revenue`) and a tuple for each client may be much more useful. This can be done through a “flattening” operation in the original view. Said process is described below.
To carry out a flattening operation on a view the option ‘Toolbox-Advanced operations-Flatten’ must be selected on the menu on the left side. Drag&drop the view to be flattened over the work space. Having done this, the schema of the input view appears on the left part of the work space. On the right part the schema of the output view appears (initially both schemas will be equal).

The next step consists of going to the input view over the array-type field to be flattened, pressing the right button on the mouse and selecting the ‘Flatten array’ option (see Figure 39).
Figure 39 Flattening the array of objects returned by \texttt{getAverageMonthlyRevenue}

The output view will then show the flattened schema (see Figure 40).
In each “flattening” operation it is possible to select just one array. To flatten more than one this operation must be repeated for each.

As usual, the name of the resulting flattened view or of its attributes can be changed. To delete attributes, the checkboxes next to the attributes that are meant to be deleted must be checked, and then the ‘Remove Selected Field’ button must be pressed. The checkbox shown in the table header allows selecting or deselecting all attributes, New derived attributes may also be added to the resulting view using the ‘New Field’ button (see section 6.4.6.1).

Finally, pressing the ‘ok’ button creates the new view obtained as a result of the flattening operation (see Figure 41).
6.4.8 Visualization in tree view mode

Using the 'Tree View' button on the screen that displays the schema of a view allows the view to be visualized in tree mode.

This visualization mode graphically displays the successive levels of views that have been composed to construct this view. For example, Figure 42 shows the tree of the view incidences_sales.

Click on any of the tree views to access the page showing its schema.

Click on the tree nodes representing combining operations (joins, unions, selections, etc.) to display their main properties. For example, in the case of selection operations the condition used to create it will be displayed.
6.4.9 Executing views and using VQL Shell

The 'Execute' button of the screen that shows the schema of a view allows the tuples of same to be obtained. The system will execute a default 'select * from viewname' query, where viewname is the name of the view. However, the tool also offers the option of accessing the conditions editor before execution to create a view selection condition (see section 6.3.4.1).

The option “Limit Rows” allows to limit the number of rows returned by the query execution.

Any query execution can be cancelled at any moment. If results are not being shown yet, cancellation can be achieved by pressing the '[X]' button of the 'Please Wait' dialog which can be seen until the first batch of results is shown. Once the query results are being shown, the execution can be stopped by pressing the 'Stop' button.

Figure 43 shows the result of executing the view incidences_sales.

The "View Execution Trace" button allows for access to the query execution trace (see section 6.4.10). If the "Limit Rows" options is used, the trace will not be available.
If the response to the query execution contains a compound field (registers or arrays of registers), double click on the field to navigate its subcomponents. For example, Figure 44 shows the result of executing the base view `average_revenue_array` (described in section 6.4.7). This view returns as a result one sole tuple that contains a field of the type array of registers. Each register contains the `taxId` and the average monthly sales of a client. Double clicking on the field '[Array]' provides access to its contents (see Figure 45). The 'Back' button can be used in this screen to return to the higher level.
Figure 44 Result comprised of one sole tuple with a field of the type array of registers
When constructing queries to a view, it is important to remember that their query conditions should fulfill the view’s mandatory capabilities (see section 6.6.3).

6.4.9.1 Using VQLShell

The tool’s VQL_SHELL tab provides access to a VQL interpreter that allows statements and queries written in this language to be sent to the server. It is possible to write one or more statements separated by ‘;’. The statements will be run on pressing the ‘Execute’ button or on pressing the key combination CTRL+ENTER. Any query execution can be stopped at any moment by pressing the ‘Stop’ button.

The option “Limit Rows” allows to limit the number of rows returned by the query execution.

The “Load” button allows for a VQL script contained in a file to be loaded. By pressing this button and choosing the ‘verbose output’ option, the tool will output each command and the result of its execution. It also allows to save the output in a file.

The “View Execution Trace” button allows for the execution trace of the last statement to be viewed (this option is not available if the “Limit Rows” option has been used). This button is enabled if the statement is run with the TRACE clause (see section 6.4.10).

The ‘Save Output’ button saves the output of the last executed statement in a file.

The console includes a “command log” that saves the last VQL statements run. There are two ways of browsing the command log:
• Using the selectable ‘Commands log’. By opening this selectable you will access a list of the last statements run and one can be selected.
• Using the keys F1 (back to previous command) and F2 (forward to next command).

It is also possible to use the view and base relation drop-down menu on the left to speed up writing VQL statements:

• Click on a view with the left-hand button of the mouse and its name will appear in the text box used to write the VQL statement.
• Click with the right-hand button and SELECT * FROM <view name> will appear in the text box.
• Click with the central button and DESC VQL <view name> will appear in the text box.

6.4.10 Execution Trace of a Statement

After a statement has been run, whether it be via the VQL Shell or the execution screen of a view, a trace of it can be accessed. The trace will only be available if the “Limit Rows” option has not been selected.

The trace of a statement provides a detailed examination of its execution plan. This plan is displayed to the administrator in the form of a tree, where each node represents an intermediate view involved in the execution of a query or access to a source via a wrapper.

The most relevant parameters are displayed for each node on the query execution tree. These parameters include:

• **Node type.** If the node is a view, this indicates the type of view (base view, union, join, projection, etc.). If it is an access to a source (wrapper), this indicates the type of source (JDBC, Web Service, Web, etc.).
• **Execution time.** Time spent completely executing the node and all its children.
• **Start time.** The exact moment at which node processing begins in the execution plan.
• **End of query time.** The exact moment at which node processing (and that of all its children) ends in the execution plan.
• **Time obtaining the first tuple of results.** Time spent until the node receives the first tuple to be processed.
• **Number of tuples processed.** Number of tuples processed by the node.
• **Status.** This indicates whether the node was correctly executed or whether an error occurred.
• **Advanced parameters.** These provide further details on each node type. For example:
  o In the case of wrapper-type nodes, the exact sub-queries executed on each data source and the connection data used to access each one are indicated.
  o For each view-type node, whether the cache has been used, whether swapping has been necessary and whether there are rewrite rules associated with it, etc. are indicated (for more information, please read the VQL Advanced Guide [VQL]).

As an example, Figure 46 shows part of the graphic trace of the execution of an inci\_dences view (created in section 6.4.1).

One of the main utilities of the trace function is debugging in the event of error conditions arising. In this case, the nodes to have caused an error during the execution will be marked red. By clicking on the nodes causing the error, the exact cause can be accessed.
For example, Figure 47 shows the execution trace of a join view where one of the data sources is not running. More precisely, the trace shown corresponds with a join view created over the incidences view (which was created in section 6.4.1) and the flat_revenue view (which was created in section 6.4.7), when the Web Service flat_revenue comes from is not running.
6.4.11 Replacement of a View Definition

It is sometimes necessary to modify a DataPort view. There are several reasons for this: due to changes in the sources requiring the updating of an existing view or to the system administrator being forced to exchange one source for another (e.g. because the data has been migrated to another system or the decision has been made to use a redundant repository).

In these cases, it must be possible to easily modify the affected view without having to regenerate the entire derived view system using the modified view. DataPort allows for this operation to be carried out very simply by creating the new view and giving it the same name as the required view. After requesting confirmation from the user (see Figure 48), DataPort will delete the previous view and update the derived views affected by the change (schemes and query capabilities are recalculated and any unnecessary elements are deleted. In the case of a change in the base view, on a different source type, the old wrapper is deleted if it is no longer used by any other view).

Where, due to the change in view, the query capacities (see section 6.6.5) of some derived views have been altered (e.g. due to the addition of another field or a query restriction that did not previously exist), DataPort will update the query capacities and restrictions of the derived views wherever possible. Where the changes introduced are incoherent in relation to the derived views (e.g. on updating a view taking part in a join operation in which any of the join attributes disappears), these will appear on the graphic interface with the symbol to the left of the name.
6.5 OTHER TYPES OF SOURCES

This section describes the process of importing data sources of the type ODBC, XML, Web, Aracne, Google Mini, delimited and CUSTOM files, as well as the process of creating base relations for same. A special case in the creation of base relations of data sources of the type JDBC/ODBC is also described in detail: creating a base relation using an SQL query “pattern”. Lastly, section 6.5.10.1 describes how to configure the configuration properties for a data source.

During the process of adding XML-type sources and delimited files it is necessary to specify paths to various documents. The available path formats in Virtual DataPort are described in section 6.5.1.

The other subsections are responsible for importing each of the source types mentioned respectively.

6.5.1 Path types in Virtual DataPort

During the process of adding XML-type sources and delimited files, paths must be specified to data files. Denodo Virtual DataPort supports different types of paths to access these documents.

The available path types are described below:

- Local path. Allows a file available in the local files system to be indicated.
- Client http path. Provides access to the file through an http request. It is required to specify: the http connection method to be used (GET/POST) and the URL to be invoked. If the http server requires authentication, the user identifier and password must be provided. If the access is realized through a Proxy, the host name and port where the Proxy is executed must be provided. If it is an authenticated proxy, the
user identifier and a valid password must be also specified. It is also possible to use the default http proxy configuration (see section 5.1.5) by clicking the option “Default”.

- FTP path. Provides access to a file available in an FTP server. The parameters required to configure this path are the url to the server and the user and password identifier to be used.

The route-type configuration screen includes a checkbox called ‘Test Connection’, which, if checked, runs a test connection to the specified route to check whether it is accessible for the administration tool or not.

Besides, in delimited file datasources, there are some additional parameters to specify the charset encoding of the file. If the ‘Autodetect encoding’ checkbox is selected, DataPort will try to automatically detect the file encoding through a test connection. If the ‘Autodetect encoding’ checkbox is de-selected, it is possible to choose the desired encoding from the options in the ‘Charset Encoding’ drop-down list.

6.5.1.1 Paths with interpolation variables

The paths to access the XML source data files and delimited files can be parameterized depending on the query made using interpolation variables (see ‘Query Run Context and Interpolation Strings’ in the Advanced VQL Guide [VQL]).

If the path specified includes interpolation variables, an intermediate step (see Figure 49) will be required during the base relations creation process, in which DataPort will request valid values for them. This is necessary so that DataPort can access a data file and obtain the metadata for creating the base relation.

![Figure 49 Providing input values to interpolation variables](image)

Each interpolation variable must be associated to a base relation attribute generated (see Advanced VQL Guide [VQL] for more details). If the specified data file includes a field with the same name as the variable, then the variable will
be associated with the base relation attribute corresponding to this field. Where the SQL query specified returns no field with the same name as the variable, DataPort will add a new attribute to the base relation with the name of the variable.

**NOTE:** When interpolation variables are used in the path description, the "\" carácter acts as escape carácter for the special characters which are used to indicate variable names ("@", 
, ){). In Windows systems, this same character is used to specify file paths, causing some ambiguous situations. For example, the system could interpret that the path "c:\tmp\@{var}" references a file named 'tmp@{var}' in the 'c:\' directory, because the second ", due to its appearance just before '@', is considered an escape carácter. However, the "c:\tmp\\@{var}" path would reference a file which name is related to the value of the var variable and it is found at "c:\tmp".

### 6.5.2 ODBC sources

DataPort provides access to ODBC sources through the use of a JDBC/ODBC bridge. The process for creating a new data source of the type ODBC is initiated by accessing the 'Data Sources – Projects – ProjectName – Relational DB – ODBC' tab and clicking the option 'New' (ProjectName represents the name of the project to which the new source is to be added). This provides access to the screen, where a new ODBC source is created (see Figure 50).

The following data are requested in this screen:

- **Name by which the Datasource will be known in Virtual DataPort.**
- **Database Adapter.** In this selectable it is posible to choose one of the DBMSs DataPort includes a specific adapter for. A "Generic" adapter is also included for ODBC-accessible DBMSs without an specific adapter. Each specific adapter implements several optimizations for its target DBMS, so they should be used whenever possible. If the "Choose automatically" option is selected (recommended), DataPort will discover the suitable adapter by connecting to the DBMS.
- **DSN (Data Source Name).** Source name of the ODBC data to be used.
- **User login to access the external database.**
- **User password to access the external database.**
- If the ‘Choose automatically’ option is marked (recommended), the system will connect to the DBMS to discover the suitable adapter. On marking this option the ‘Test Connection’ option is automatically selected.
- **Finally, if the ‘Test Connection’ option is marked, the system will connect to the source to verify it can be accessed from the DataPort server.**

Optionally, the “Connections Pool Configuration” option may also be accessed to configure various parameters of the connections pool that Virtual DataPort will use to optimize access to the external database. The available parameters are the same as for the case of JDBC sources (see section 6.3.1).
Once the ODBC source has been imported, the process for creating base relations is the same as that for JDBC (see section 6.3.3).

### 6.5.3 XML sources

To create a new data source representing an XML document, access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click on the ‘New’ option that appears beside the list of current XML sources. This provides access to the creation screen for a new XML data source (see Figure 51). The following data are requested in this screen:

- **Name** to be given to the data source in Virtual DataPort.
- **Path** to the XML file that contains the required data. This path can be parameterized according to the query made using *interpolation variables* (see section 6.5.1.1).
- **Path** to the file containing the specification of the schema of the XML document (optional). The schema may be described using the DTD specification or the XML Schema specification. If it is not specified (marking the ‘None’ option), Virtual DataPort will seek to infer an appropriate schema by analyzing the XML document structure containing the data.

The path formats available are described in detail in section 6.5.1.
By pressing ok the new datasource will be created and access given to the XML source description screen (see Figure 52).

On clicking the ‘Create Base View’ button, the schema of a base relation associated with the source is automatically displayed (except if the path to the data file includes interpolation variables, in which case there will be an intermediate step. See section 6.5.1.1). The name of the base relation and the name and type of each of the constituent attributes may be changed at this stage. The project to which the new view is to be allocated can also be specified using the selectable ‘Target Project’.
By clicking on a base relation we have created we can access the page which displays its schema in Virtual DataPort.

Once the base relation has been created, a projection view of it should be used to modify its schema to eliminate attributes or change the name of same (see section 6.4.6). Pressing the ‘Edit’ button provides direct access to the screen in which a projection view of the base relation is created.

6.5.4 Web Sources

Web data sources allow the Web wrappers created through Denodo ITPilot [ITPILOT] to be used. This type of source can only be imported and used, where Denodo ITPilot is installed in the same path as the DataPort server.

To create a new data source of this type access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click the option ‘New’ that appears beside the current list of ITPilot sources. Access will then be given to the creation screen (see Figure 53). A file is requested in this screen containing the VQL statements for creating the wrapper. This file is generated directly by the Denodo ITPilot generation environment.
By pressing ok the new Web source is added. Clicking on it gives access to a screen from which a base relation can be created from same. The name of the base relation and the type of each of the constituent attributes may be changed in this screen. The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’.

By clicking on a base relation we have created we can access the page which displays its schema in Virtual DataPort. Once the base relation has been created, a projection view of it should be used to modify its schema to eliminate attributes or change the name of same (see section 6.4.6). Pressing the ‘Edit’ button provides direct access to the screen in which a projection view of the base relation is created.

6.5.5 Delimited file sources

This type of source allows Denodo Virtual DataPort to access the data contained in flat files in CSV (Comma Separated Values) format or similar.
To create a new data source of this type access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click the option ‘New’ that appears beside the current list of delimited file sources. Access is then given to the screen, where a new data source of the type delimited files can be created (see Figure 54). The following data are requested in this screen:

- Name to be given to the data source in Virtual DataPort.
- Path to the file that contains the required data. The formats of the available paths are described in detail in section 6.5.1. The path can be parameterized according to the query made using interpolation variables (see section 6.5.1.1).
- Column delimiter. Indicates the character string to be used as a separator between fields of the same tuple.
- Tuple end delimiter. Indicates the character string to be used to mark the end of a tuple. The string ‘\n’, which identifies a carriage return, will be used by default.
- Begin of Data Zone Delimiter: A JAVA regular expression identifying the position in the file where the system must start searching for tuples (or searching for the header if the ‘header’ option was checked). If not value is specified, the search will start at the beginning of the file.
- Include Begin Delimiter as Data. If this option is marked the text matching with the begin regular expression will be considered as part of the search space.
- End of Data Zone Delimiter: A JAVA regular expression identifying the position in the file where the system must stop searching for tuples. If not value is specified, the search will continue until the end of the file.
- Include End Delimiter As Data. If this option is marked the text matching with the End regular expression will be considered as part of the search space.
- Header. This verification box indicates, if the first line with data of the file contains the names of the fields. If this is the case, said names will be used to create the attributes of the base relation for DataPort.

Pressing ok the new data source is created and access is given to the delimited files source description screen (see Figure 55). Pressing ‘Create Base View’ automatically creates the schema of a base relation associated with the
source (except if the path to the data file includes interpolation variables, in which case there will be an intermediate step. See section 6.5.1.1). The name of the base relation and the name and type of each of the constituent attributes may be changed at this stage. The project to which the new view is to be allocated can also be specified using the selectable ‘Target Project’.

By clicking on a base relation we have created we can access the page which displays its schema in Virtual DataPort. Once the base relation has been created, a projection view of it should be used to modify its schema to eliminate attributes or change the name of same (see section 6.4.6). Pressing the ‘Edit’ button provides direct access to the screen in which a projection view of the base relation is created.

6.5.6 Google Mini Sources

Denodo Virtual DataPort allows for Web search engines built using the Google Mini software to be used as source [GMINI].

To create a new data source representing a Google Mini search engine, access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click on the ‘New’ option beside the list of current Google Mini sources. This provides access to the creation screen for a new data source of this type (see Figure 56). The following data is requested on this screen:

- Name to be given to the data source in Virtual DataPort.
- Name of the machine in which the search engine is accessible. This may be a remote machine accessible on the Internet (e.g. “search.acme.com”).
- Port on which the Google Mini service is run. This port is often 80.
Proxy Configuration. If the access is realized through a proxy, the host name and port where the Proxy is executed must be provided. If it is an authenticated proxy, the user identifier and a valid password must be also specified. It is also possible to use the default http proxy configuration (see section 5.1.5) by clicking the option "Default".

By clicking ok, the new data source is created and access is given to the Google Mini source description screen (see Figure 57), where the fields available in the search engine are displayed. These fields are fixed for all Google Mini search engines.
By clicking on 'Create Base View', the schema of a base relation associated with the source is automatically created (see Figure 58).

**Figure 57** Description of the Google Mini source

By clicking on 'Create Base View', the schema of a base relation associated with the source is automatically created (see Figure 58).
A value can be specified for the following parameters at the top of the screen:

- **Site Collections.** This parameter is mandatory. It specifies, within the Google Mini server, the collections on which to make the search. The collections are created by the Google Mini server administrator. Its name is upper/lower case-sensitive. It is possible to specify several collections separated by commas. In this case, the search will be made on all of them. Where an external server is accessed, the collection to be searched can normally be obtained by examining the value of the site parameter on its invocations.
- **Client:** This parameter is optional. It identifies the client making the queries. The Google Mini server can be configured to behave in a different manner, depending on the client to have issued the query.
- **Languages:** This parameter is optional. If specified, only documents in the specified language will be returned. The language must be a value of those listed in the Google Mini documentation [GSEARCHLANG].
- **Number of Key Match:** This parameter is optional. Google Mini allows for the administrator to manually determine the priority of the pages, when the results of a search are displayed. This parameter receives an integer value of between 0 and 5, where 5 is the maximum priority. If this value is established, the searches made on this base relation will only return the pages with the priority specified or over.

The value of these parameters can be subsequently modified on the advanced configuration screen of the base view created (see section 6.6.7).

The name of the base relation can also be modified and the name and type of each of the non-required attributes deleted on the base view creation screen. The contents of each one of the attributes is as follows:

- **TITLE.** Title generated by Google Mini for the document.
- **SUMMARY.** Summary generated by Google Mini for the document.
- **URL.** Document URL.
- **MIMETYPE.** MIME type of the document.
• RATING. Priority assigned by the Google Mini administrator for the document. This may take values of between 0 and 5, where 5 is the maximum priority.
• MAXDOCS. Attribute added by DataPort to restrict the maximum number of results returned by a search.
• METAS. Array-type compound record attribute (see Advanced VQL Guide [VQL]) that contains the metatags for the document. Each record has two fields to indicate the name of the metatag (metakey) and its value (metavalue).
• CONTENT. Contents of the document. This is the field normally used for searches.
• SITE. This allows for the documents returned to be restricted to those belonging to a certain domain (e.g. `acme.com`).
• FILETYPE. Extension of the document file.

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’.

The base view is created just by clicking ‘ok’. Once the schema has been accepted, the new base relation may be viewed on the left of the screen.

The most common way of querying the base relations built from Google Mini sources is using the **CONTAINS** operator, which runs complex Boolean searches on indexed textual data (see Advanced VQL Guide [VQL]).

### 6.5.7 Aracne Sources

Denodo Virtual DataPort allows for indexes built using Denodo Aracne 4.0 or above [ARCN] to be used as a source.

To create a new data source representing an Aracne index, access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click on the ‘New’ option that appears beside the list of current Aracne sources. This provides access to the creation screen for a new Aracne data source (see Figure 59). The following data are requested on this screen:

• Name to be given to the data source in Virtual DataPort.
• Name of the machine in which the search engine is accessible.
• Port on which the Denodo Aracne search server is run. This port is 4000 in the Aracne default installation.
By clicking ok, the new data source is created and access is given to the Aracne source description screen (see Figure 60). The fields available are displayed for each content index in the Aracne server.

**Figure 59** Creating an Aracne data source
By clicking on 'Create Base View', the schema of a base relation associated with the index selected is automatically created (see Figure 61). The name of the base relation and the name and type of each of the constituent attributes may be changed and any non-required attributes deleted at this stage.

All pregenerated base views using Aracne indexes have a series of fixed attributes, although they may also include specific attributes, where the initial indexes have them. Below is a description of the fixed attributes (see the Aracne Administrator Guide [ARCN] for further details):

- **TASK**: Name of the Aracne task that obtained and indexed this document.
- **PUBDATE**: Document publication date. This only appears in the event of the index containing RSS-type documents.
- **TITLE**: Title generated by Aracne for the document.
- **ANCHORTEXT**: Where a document obtained by Aracne using a Web crawling process contains the link text used to access the document.
- **SUMMARY**: Summary generated by Aracne for the document.
- **URL**: In the case of documents obtained over the Web page, this contains the original document URL. This corresponds to the link field value of the RSS item in RSS documents. In the case of documents obtained from a local file system, this contains the path to it. In the case of documents obtained from an e-mail server, it contains the name of the e-mail server and the name of the account to which the e-mail belongs.
- **IDENTIFIER**: Standardized URL.
- **CONTENT**: "Useful" contents of the document generated by Aracne. See the Aracne Administrator Guide [ARCN] for further details.
- **DESCRIPTION**: This only appears in the event of the index containing RSS-type documents. In this case, it takes the value of the DESCRIPTION element from the RSS document.
- **MODIFIED**: Date on which the document in the index was last modified.
• SEARCHABLECONTENT. Attribute added by DataPort that represents the concatenation of the contents of the main fixed textual fields of the index (title, summary, contents, anchortext, etc.) and the specific fields that the index may contain. This is the field on which searches are normally made.
• LEVEL. Crawling depth level at which the document was obtained.
• TYPE. Content type: html, pdf, rss, etc.
• TITLEXML. Title of the document in XML with information on the view structure of the contents (paragraphs). This field is used to visually represent the title and not for searches. See the Aracne Administrator Guide [ARCN] for a description of the format used.
• SUMMARYXML. Summary of the document in XML with information on the view structure of the contents (paragraphs). This field is used to visually represent the summary and not for searches. See the Aracne Administrator Guide [ARCN] for a description of the format used.
• PATH. Where the Aracne server saves a local copy to the document, this contains the path to it.
• SCORE. Indication of the relative relevance of the document for the query. The results of a search are normally returned in decreasing order by SCORE.
• MAXDOCS. Attribute added by DataPort to restrict the maximum number of results returned by a search.
• CATEGORIES. This only appears in the event of the index containing RSS-type documents that contain a CATEGORIES element. In this case, it takes the value of this element from the RSS document.

Alongside each attribute of the base view proposed is the “Create Main Terms” option. The use of this option is explained in subsection 6.5.7.1.

The project to which the new view is to be allocated can be specified using the selectable ‘Target Project’ appearing below the view schema.
The base view is created just by clicking ‘ok’. If the user wishes to create a base view and then import another one from the same DataSource, he/she can press the ‘OK and import new view’ button, which will create the view and will go back to the screen where the Aracne server’s available indexes are shown. Once the schema has been accepted, the new base relation may be viewed on the left of the screen.

The most common way of querying the base relations built from Aracne sources is using the \texttt{CONTAINS} operator, which runs complex Boolean searches on indexed textual data (see \textit{Advanced VQL Guide} [VQL]).

6.5.7.1 Accessing the most relevant terms of a document

Denodo Aracne is capable of automatically generating the most relevant words of a document or a field thereof, according to the TFIDF (Term Frequency Inverse Document Frequency) relevance measurement. These terms can be accessed via DataPort, as described in this section.

The most relevant terms are accessed as new attributes in the DataPort base view. To create a new attribute in the base view containing the most relevant terms of the value of a certain document field, click on the ‘Create main terms’ option alongside the required field.

For example, if you want to add a new attribute containing the most relevant terms of the \texttt{SEARCHABLECONTENT} index field, click on the ‘Create main terms’ option alongside it. Figure 62 shows this new attribute (known as \texttt{SEARCHABLECONTENT\_MAIN\_TERM}).

\begin{figure}[h]
\centering
\includegraphics[width=\textwidth]{figure62.png}
\caption{Adding an attribute with the most relevant terms of the \texttt{SEARCHABLECONTENT} field}
\end{figure}
The new attribute SEARCHABLECONTENT_MAIN_TERM will be an array-type compound record attribute (see Advanced VQL Guide [VQL]). Each record contains two fields:

- The relevant term. This field takes the name of the index field by default, adding the suffix _TERM (in this case, therefore, the name will be SEARCHABLECONTENT_TERM). However, as can be seen in the figure, the name of the field can be changed.
- Its position in the list of the most relevant. This field takes the name of the index field by default, adding the suffix _SCORE (in this case, therefore, the name will be SEARCHABLECONTENT_SCORE). However, as can be seen in the figure, the name of the field can be changed. The most relevant term takes position 1.

In the new attribute creation process, it is possible to specify two parameters:

- Number of main terms. Maximum number of relevant terms to be included for each document.
- Filter main terms words. List of "usual words" (separated by commas) that must not appear among the most relevant terms for this field. Where Aracne generates any of those appearing in this list among the most relevant terms for the attribute contents, this would be eliminated from the list of relevant terms. It is important to note that only usual words specific to the application must be specified. The usual words in the language used such as articles, pronouns, etc. (commonly known as "stopwords") are already eliminated by Denodo Aracne. The list of usual words may be contained in a file. The file specified must be a text file, where the words will be separated by commas.

Furthermore, the bottom of the base view creation screen will display the ‘General Words to Filter Main Terms’ parameter that indicates a list of usual words common to all fields in the base view. Once again, you do not have to worry about specifying usual words in the language used such as articles, pronouns, etc. (commonly known as "stopwords"), as they are already eliminated by Denodo Aracne.

The value of these parameters can be subsequently modified on the advanced configuration screen of the base view created (see section 6.6.7).

### 6.5.8 LDAP sources

LDAP-type data sources can register an LDAP server in DataPort. The authentication of a DataPort user can be delegated to any of the LDAP servers registered (see section 8.3.3 for further details on the creation of users in DataPort).

To create a new data source of this type, access the “Data Sources” tab and, after selecting the project to which the new source is to be added, click on the ‘New’ option that appears beside the list of current LDAP sources. Access will then be given to the creation screen (see Figure 63). The following data are requested on this screen:

- Name to be given to the data source in Virtual DataPort.
- Path to the LDAP server. For example: `ldap://acmehost:389`, where `acmehost` is the name of the machine in which the server is housed and, in this case, is run in port 389.
- Optionally, it is possible to check if the specified LDAP server is available by checking the ‘Test Connection’ option.
On clicking on 'ok', access will be given to the screen displaying the properties of the new data source (see Figure 64).
Once the new data source has been created, it is available to create new users (see section 8.3.3).

6.5.9 CUSTOM Sources

CUSTOM-type data sources (also called MY) allow ad-hoc wrappers to be included in Denodo DataPort that have been specifically created to access a specific proprietary source. See the VQL Advanced Guide to find out how to develop CUSTOM wrappers.

DataPort includes a series of sample CUSTOM wrappers in the path `$DENODO_HOME/samples/vdp/wrappersCustom`. The README file in this path contains instructions on how to compile, install and use them. The screenshots displayed in this section use the sample CUSTOM wrapper data as an example `com.denodo.vdp.demo.wrapper.custom.MetaPayRollWrapper`.

**Figure 64** LDAP data source properties
To create a new data source of this type go to the “Data Sources” tab and, after selecting the project to which the new source is to be added, click on the ‘new’ option that appears beside the current list of sources. You then access the screen in which new CUSTOM data sources are created (see Figure 65). The following data are requested in this screen:

- Name to be given to the data source in Virtual DataPort.
- Full name of the JAVA class that implements the CUSTOM-type wrapper. Said class should be found in the application classpath. For this the class may be included from the path \$DENODO_HOME/extensions/dev/target/classes. The class may also be placed in any other path, wherever it is included in the value of the environment variable \$DENODO_EXTERNAL_CLASSPATH.
- Classpath. If required, an additional Classpath to be used by CUSTOM data source implementation can be specified.

Where the examples of CUSTOM wrappers included with DataPort have been compiled and installed, follow the instructions and click on ok to create the new data source (see Figure 66).
Click on the "Create Base View" button to create a base view associated with the new data source. The CUSTOM wrapper used to implement access to the source can accept input parameters (see the Advanced VQL Guide [VQL]). In this case, the user was requested the value of these parameters to be used for this base view (see Figure 67).
By clicking on ok, the schema of a base relation associated with the data source is automatically created (see Figure 68).
The project to which the new view is to be allocated can also be specified using the selectable ‘Target Project’

By clicking on a base relation we have created we can access the page which displays its schema in Virtual DataPort

Once the base relation has been created, a projection view of it should be used to modify its schema to eliminate attributes or change the name of same (see section 6.4.6). Pressing the ‘Edit’ button provides direct access to the screen in which a projection view of the base relation is created.

### 6.5.10 JDBC/ODBC sources through SQL Query

A special case in the creation of base relations for sources of the type JDBC/ODBC arises, when instead of using a table as a “template” for creating the base relation, we want to use an arbitrary SQL query to the source for said purpose. This can be useful in two basic scenarios:

- This method of creating base relations must be used whenever it is required to execute either a database stored procedure, or a query that uses a stored procedure, in order to obtain the desired results.
- If the administrator has already defined a complex SQL query against the source database to return the required data to the base relation, then it can be faster to create the base relation by just specifying such query, than importing all base tables used in the query to DataPort, and then creating the SQL query’s equivalent combination view.

To create a SQL Query-type base relation, the ‘Create Base View From Query’ button on the JDBC or ODBC data source detail screen should be used to access the screen shown in Figure 89.
Once the name has been specified for the base relation and the required SQL query, pressing 'ok' creates the schema of the new base relation. Its schema can be edited in the same way as in the step before creating a base relation.

The SQL query used to create the base relation may also use “interpolation variables” (see Advanced VQL Guide [VQL]), which allow the SQL query made on the source to be parameterized according to the specified query conditions. In this case, when ‘OK’ is pressed to create the base relation, the user is presented with a wizard (see Figure 70) requesting values for the variables used so that DataPort can execute the query to obtain the metadata required to create the base relation.

Each interpolation variable must be related with an attribute belonging to the generated base relation, so that the variable can obtain its value at run time (see Advanced VQL Guide [VQL] for more details). If the SQL query used to access the data source returns any field with the same name as the variable, then the variable will be associated to the corresponding base relation attribute.

In case the SQL query does not return any field with the same name as the variable, DataPort will add a new attribute to the base relation, with the same name.
6.5.10.1 Using the WHEREEXPRESSION variable

As commented in the previous section, the SQL Query used to create the base relation may use “interpolation variables” (see the VQL Advanced Guide [VQL]), that can parameterize the SQL Query executed against the source according to the query conditions specified in DataPort.

There exists a predefined interpolation variable called WHEREEXPRESSION that usually simplifies the process of creating base relations by using the SQL Query method.

Besides, the use of WHEREEXPRESSION can also help optimize the execution. Specifically, if a join view is using the NESTED execution method (please see the “Query Optimization” section in the VQL Advanced Guide [VQL]), and the view that acts as second relation in the join operation is of the SQL Query type, then it is highly advisable to create that view by using the WHEREEXPRESSION variable, because that allows DataPort to apply optimization techniques that would not be possible otherwise.

The use of WHEREEXPRESSION follows. The WHEREEXPRESSION variable might be used in the SQL query that is specified, as substitute of all or part of the WHERE clause in the query. At run time, DataPort will substitute the variable by a valid query condition built from the query conditions received by the base relation. For example, let’s suppose a base relation called VIEW1 is created by using the following SQL query:

```sql
SELECT StorProc(FIELD1), FIELD2, FIELD3, FIELD4 ALIAS4
FROM TABLE1
WHERE @WHEREEXPRESSION
```
Please notice that the query uses a stored procedure in the SELECT clause, thus making it necessary to create the base relation by using a SQL query. As explained in the previous section, after pressing ‘ok’, the user will be shown a wizard (see Figure 70), asking him/her to provide values for the variables used, so that DataPort can execute a query to obtain the metadata required for creating the base relation. In this example, any valid query condition could be used as value for WHEREEXPRESSION like, for instance:

\[ \text{FIELD2='f2' AND FIELD4='f4'} \]

Once DataPort has obtained the required metadata by executing the sample query, it will show the list of base view attributes that will be generated (see Figure 71). At that moment, the user may specify information about any aliases used in the query SELECT clause. For instance, in the previously commented query, the ALIAS4 alias is defined, which is associated to the FIELD4 field. Therefore, the user must specify the FIELD4 value for the attribute named ALIAS4.

When the alias information is specified, and after pressing ‘ok’, the base relation creation process will proceed.

In the example, let’s consider what would happen if, once the VIEW1 view has been created, the following VQL query is executed (NOTE: in the example, it is assumed that the user has not modified the attribute names when creating the base relation, and, therefore, they match the ones specified in the SQL query used to create the base relation):

\[ \text{SELECT * FROM VIEW1 WHERE FIELD2='f2' AND ALIAS4='f4'} \]

In this case, DataPort will substitute the WHEREEXPRESSION variable at run time, by the value required to execute the equivalent query on the original database. In this case:

\[ \text{SELECT StorProc(FIELD1)AS ALIAS1, FIELD2, FIELD3, FIELD4 AS ALIAS4 FROM TABLE1} \]
WHERE FIELD2='f2' AND FIELD4='f4'

6.5.11 Data Source Configuration Properties

Virtual DataPort maintains properties for each data source and base view, which allow for specific characteristics of the underlying sources to be configured such as their distributed transaction support capacity or whether inserting operations are permitted. This function allows the system administrator to optimally configure the characteristics of each data source or base view and, as a result, its possibilities for combining and running.

The properties of each data source can be configured by selecting the required data source and pressing the 'Edit' button before clicking on the "Source Configuration" link. This link only appears in data sources with configurable properties (JDBC and ODBC sources in distribution 3.5). The configurable properties are as follows:

- **Delegate All Operators** (DS: JDBC, ODBC). This indicates whether the source allows for all operators to be delegated. The value is "false" by default.
- **Delegate Array Literal** (DS: JDBC, ODBC). This indicates whether the source allows for array-type compound constants to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Compound Field Projection** (DS: JDBC, ODBC). This indicates whether the source allows projections on compound fields to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate GROUP BY** (DS: JDBC, ODBC). This indicates whether the source allows the GROUP BY clause to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate HAVING clause** (DS: JDBC, ODBC). This indicates whether the source allows the HAVING clause to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Inner Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Inner Join operator to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Join operator to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Left Function** (DS: JDBC, ODBC). This indicates whether the source allows for conditions with functions on the left part to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Left Literal** (DS: JDBC, ODBC). This indicates whether the source allows for conditions with constants on the left part to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Natural Outer Join** (DS: JDBC, ODBC). This indicates whether the source allows for conditions with the Natural Outer Join operator to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate NOT Condition** (DS: JDBC, ODBC). This indicates whether the source allows the NOT condition to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate OR Condition** (DS: JDBC, ODBC). This indicates whether the source allows for the OR condition to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate ORDER BY** (DS: JDBC, ODBC). This indicates whether the source allows for the ORDER BY clause to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Projection** (DS: JDBC, ODBC). This indicates whether the source allows for projections to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Register Literal** (DS: JDBC, ODBC). This indicates whether the source allows for register-type compound constants to be delegated. The value is "false" by default for JDBC and ODBC sources.
- **Delegate Right Function** (DS: JDBC, ODBC). This indicates whether the source allows for conditions with functions on the right part to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Right Literal** (DS: JDBC, ODBC). This indicates whether the source allows for conditions with constants on the right part to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate Selection** (DS: JDBC, ODBC). This indicates whether the source allows for conditions to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Delegate UNION** (DS: JDBC, ODBC). This indicates whether the source allows for the union operator to be delegated. The value is "true" by default for JDBC and ODBC sources.
- **Supports Modifier in Aggregate Function** (DS: JDBC, ODBC). This indicates whether the source supports DISTINCT/ALL modifiers in aggregate functions.
Virtual DataPort 4.1 Administrator Guide

6.6 ADVANCED CONFIGURATION OF VIEWS

The ‘Advanced’ button on the schema screen of a view provides access to its advanced configuration wizard (see Figure 72). The view cache, its query capacities, its swap to disc policy, certain characteristics of its run plan (only in derived views), its internationalization configuration (only in base views) and some specific aspects for views from some types of sources can be configured from this screen. The following subsections deal with each of these aspects.

- **Supports Branch Outer Join** (DS: JDBC, ODBC). This indicates whether the source allows for (left | right) outer join. The value is “false” by default for JDBC and ODBC sources.
- **Supports Eq Outer Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Equality Outer Join operator to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Explicit Cross Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Explicit Cross Join operator to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Full Eq Outer Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Full Equality Outer Join operator to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Full NotEq Outer Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Full Not Equality Outer Join operator to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Fusing in Using AND Natural Join** (DS: JDBC, ODBC). This indicates if the source merges the same fields when running a natural join or a join with the USING clause. The value is “false” by default for JDBC and ODBC sources.
- **Supports Join On Condition** (DS: JDBC, ODBC). This indicates whether the source allows for the Join On clause to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Natural Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Natural Join clause to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Supports Using Join** (DS: JDBC, ODBC). This indicates whether the source allows for the Using Join clause to be delegated. The value is “false” by default for JDBC and ODBC sources.
- **Delegate Aggregate Functions List** (DS: JDBC, ODBC). This indicates the aggregation functions that can be delegated. In JDBC and ODBC sources, the list is made up of the AVG, COUNT, MAX, MIN and SUM functions.
- **Delegate Scalar Functions List** (DS: JDBC, ODBC). This indicates the scalar functions that can be delegated. In JDBC and ODBC sources, the list is made up of the ABS, CEIL, CONCAT, DIV, FLOOR, GETDAY, GETHOUR, GETMINUTE, GETSECOND, GETMONTH, GETYEAR, LEN, LOG, LOWER, MOD, MULT, NOW, POWER, REPLACE, ROUND, SQRT, SUBSTR, SUM, TEXTCONSTANT, TRIM and UPPER functions.
- **Delegate Operators List** (DS: JDBC, ODBC). This indicates the operators that can be delegated. In JDBC and ODBC sources, the list is made up of the =, <=, >=, in, between, contains, containsor, like, isnull, isnotnull, istrue and isfalse operators.

Virtual DataPort has default values for some specific relational databases (MySQL, Oracle, Postgres, etc.) that may vary in relation to those described above.
Internationalization configuration

The internationalization configuration of a base relation specifies aspects such as time zones, languages or currencies. For example, the currency to be used in the values of the type Money is specified by this property.

DataPort includes typical internationalization configurations and also allows new configurations to be created ad hoc. See the Advanced VQL Guide [VQL] for more information.

Configuring the cache of a view

The following parameters may be used for the view cache:

- **Time to live**. Indicates the time to live (in seconds) for the data stored in the view cache. If the ‘Default’ checkbox is checked, the default value defined in the server configuration panel will be used (see section 5.1.3).
- **Cache on/off/post**. The cache can be activated or deactivated for the view using the cache on/off selector. The option ‘post’ activates the cache, while also allowing post-processing. If the option ‘post’ is enabled, the system will detect whether a given query can be answered using another previous query, even where this is not the same as the new query. This is carried out through the use of a series of post-processing operations. For example, if the results of a previous query `select * from view where field1 = a` are in the cache and the system receives the query `select * from view where field1 = a and field2 = b`, it can be responded to on the basis of the results of the first query, applying a post-processing operation that eliminates those tuples in which `field2 = b` does not apply. If this option is disabled, the system only uses the cache, if the query received is the same as the preceding query.
Use of the post-processing technique may not be desirable, if a wrapper does not always return all results for a query made to a specific source. For example, if a wrapper that accesses a Web source returns only the first 100 results returned by the source for the query `select * from view where field1 = a`, then the result of applying the post-processing condition `(field2 = b)` to the results of the query may be different to the result obtained when executing directly on the source `select * from view where field1 = a and field2 = b`.

### 6.6.3 Swapping configuration

To avoid memory overflows, Virtual DataPort can swap the intermediate results generated during a view execution process to disk.

To enable or disable swapping for the view, use the options “Swap on” or “Swap off” (default option), respectively. Global level “swapping” must have been previously enabled in the server (see section 5.1.6).

As a view or query is being run, DataPort will swap to disk when an intermediate result produced during the execution exceeds the maximum size specified.

This size can be specified (in megabytes) using the “Max size” field. As a general rule, the specified value should be no greater than one third the memory available for the JAVA virtual machine on which the DataPort server is run. If the ‘Default’ checkbox is checked, the default value defined in the server configuration panel will be defined (see section 5.1.6).

**NOTE:** DataPort can also configure the use of “swapping” dynamically for a specific query using the `VQL CONTEXT` clause (see the Advanced `VQL` Guide [VQL]).

### 6.6.4 Editing the Execution Plan

A key aspect of query and view run optimization in DataPort is the execution strategy used for each join operation involved.

As described in section 6.4.3, it is possible to specify the required execution strategy on creating a join-type view. A execution strategy for a join operation indicates two parameters (see ‘Query Optimization’ in the Advanced `VQL` Guide [VQL] for more details):

- Run method (merge, hash or nested) and
- Order in which the input relations must be considered. The view at the top will be considered as a first relation, if the Ordered option is chosen, and as second, if Reverse Order is chosen.

In addition, it is also possible to choose the ANY option, so DataPort will choose the strategy to apply.

However, a join view can be used in defining new derived views and the optimal execution strategy may be different in each case.

**Example:** Suppose there is a join view `V3` with attributes `A`, `B` and `C`. `V3` has views `V1` (with attributes `A` and `B`) and `V2` (with attributes `B` and `C`) as input relations.

Also suppose that queries on `V3` are using the nested-loop execution method. Common optimization practice on using this execution method involves ordering the input relations so that the first is the one to provide least tuples to the join operation.

In our example, suppose that the number of input tuples to the join from `V1` is normally less than that of `V2` and, therefore, `V3` is usually queries considering `V1` as the first relation.
Now suppose that you want to create a new view $V^4$ that is a selection view on $V^3$ by applying the condition $C=c$. Also suppose that there are few tuples in $V^2$ that verify this condition. Then, on running a query on $V^4$, the most common situation will be for the join view $V^3$ to be given less tuples from $V^2$ than from $V^1$. Therefore, when $V^4$ is run, $V^2$ should be considered the first relation, unlike the general case where the opposite is sought.

To deal with these cases, in any derived view $V$ it is possible to define the preferred run strategy for each join view involved in its definition. These strategies will only be used to run queries on $V$ and will not overwrite the default execution preferences of these join views.

---

**Figure 73** Editing the Execution Plan of a view

Hence, the preferences for the execution plan of $V$ will not affect the queries made directly on the original join views or on other derived views using them. It is important to note that this can only be done when the join operation was created using *ANY* as execution strategy.

To edit the preferences on the join execution strategy for a derived view, access the advanced configuration screen and click on the ‘Edit Execution Plan’ button. The execution plan edit screen will then be displayed for the view (the execution plan edit screen is shown in Figure 73 for view *inc_grouped_by_pref_clients* created in section 6.4.5).

The screen displays the view definition tree. For each join operation involved in the definition, it is possible to mark the ‘Specify execution plan (do not inherit it)’ box to enable two selectable fields to choose the execution method and the order of the input relations required. If the ‘ANY’ option is chosen for a parameter, DataPort will use internal cost information to choose the most suitable option.

On clicking ‘ok’, you will return to the advanced configuration screen for the view. By clicking ‘ok’ again, the configuration changes made on this screen will become effective.
NOTE: DataPort can also configure the execution plan preferences for a specific query using the VQL CONTEXT clause (see the Advanced VQL Guide [VQL]).

6.6.5 Query capabilities

Some data sources such as Web sites or systems that offer a Web Service interface do not allow any query to their data, instead they present interfaces with limited query capability (e.g. HTML forms in the case of a Web site and a list of parameters in the case of a Web Service operation). For this reason, Virtual DataPort allows the administrator to explicitly specify the query capabilities of the base relations or views. Also, Virtual DataPort automatically calculates the query capabilities of the derived views according to the capabilities of its base views.

Therefore, when the advanced configuration of a view is accessed, the available options vary according to whether the view is a base relation (also called base views) or a derived view. The difference lies in that in the case of base relations the query capabilities can be modified, while in the case of derived views they can be queried but not directly modified.

The query capabilities are expressed using a list of search methods. Each search method specifies a different way to query the view (or base relation). See the next section for a detailed description of how to interpret search methods of a view (or base relation).

When a base relation is created, the system automatically assigns search methods to it. If in the majority of cases the methods automatically created do not need to be modified, this may not always be the case, particularly in sources of the type Web and Web Service.

Once in the advanced configuration view of a base relation, the default search methods generated for same may be changed easily (see Figure 72). As can be seen in the figure, a search method specifies for each attribute of the view (see the subsection below for more details):

- Its data type,
- A list of operators for which the field can be queried. ‘any’ indicates any operator. New operators may be added for the attribute using the drop-down menu in its ‘Operators’ column. Operators may also be eliminated using the buttons ‘[x]’ that appear beside each operator.
- Whether it is obligatory (‘OBL’), optional (‘OPT’) or not supported (‘NOS’) in the queries to the view.
- The multiplicity supported for the attribute in queries to the view.
- A list of the possible values for the attribute that may appear in a condition of a valid query to the view. If no value is specified, it is assumed that any is valid. To add new values simply key them into the text box that appears in the ‘Values’ column of the attribute and press the button that appears on the right. A value may be deleted using the button ‘[X]’ that appears at its side.
- Whether the attribute appears in the response of the queries made to the view.

The VQL language may be used directly to add or delete complete search methods (see the Advanced VQL Guide [VQL]).

6.6.5.1 Search Methods

Each base relation explicitly describes its query capabilities through what are known as search methods. If a relation has no search method, then no query can be made to it.

Each search method is comprised of a series of 5-tuples. Each 5-tuple represents a restriction that a specific query should comply with to be executed on the source using this search method. In this sense, a search method that has no 5-tuple allows any query (this is the case of sources that have no limitations in their query capabilities such as, for example, conventional relational databases).

The format of a 5-tuple is (attribute, operators, obligatoriness, multiplicity, possible_values) where:
- **attribute** is an attribute of the relation.

- **operators** is the group of operators that can be used in the queries to this view and with this search method. ‘ANY’ represents any operator allowed by the attribute data type.

- **obligatoriness** can have three values: ‘OBL’ indicates that the attribute should mandatorily appear in any query to the source. ‘OPC’ indicates that the attribute may or may not appear in the query (it is optional) and ‘NOS’ indicates that the queries for this attribute are not allowed in the source.

- **multiplicity** indicates how many values can be included in the source query for the given attribute and operator. If it is not possible to make queries for this attribute ("NOS" value in the obligatoriness field), the value is necessarily 0. ‘+’ indicates a number of values greater than 0 but without an upper limit.

- **possible_values** is the list of values that can be used to query the attribute. If the value “ANY” is contained in it, this means that the search range is not limited (within the range associated with the attribute data type), and the attribute can be queried about any value. If the obligatoriness field is set in the 5-tuple to the ‘NOS’ value, then it necessarily takes the value of an empty group.

*Example:* Consider the example of a virtual bookshop on the Internet the search form of which is like that shown in Figure 74.

![Figure 74 Search form for a bookshop](image)

The form obliges the user to specify a value for the TITLE attribute and gives the option to set a value for the AUTHOR attribute and for the FORMAT attribute (restricted to a group of values). Searches by title and author are searches by keyword (operator ANY). A search by exact phrase (operator =) is indicated selecting the box next to the search box of the field. For each attribute a simultaneous search is allowed using one value only. In addition to the fields TITLE, AUTHOR, and FORMAT, we can expect that the shop returns as output a PRICE attribute, which cannot be queried directly using the form.

Let us model this source as a relation \( R = \{ \text{TITLE}, \text{AUTHOR}, \text{FORMAT}, \text{PRICE} \} \) with a search method containing the 5-tuples shown in Figure 75.
Note that the first 5-tuple has the value \{like, =\} in the OPERATORS field and OBL in the OBLIGATORINESS field, this does not mean that it is obligatory to query the TITLE attribute with both operators, but that it is obligatory to query it at least with one of them. In order to have the TITLE attribute appear obligatorily in the query with both operators (this is not possible in the form in the example), this should be done with two different 5-tuples for the TITLE attribute, one for each operator:

\{ (TITLE, \{ANY\}, OBL, 1, ANY) (TITLE, \{=\}, OBL, 1, ANY) \}.

Thus, as can be seen, when you want to differentiate the treatment of a specific attribute according to the operator with which it is used, more than one 5-tuple can exist for each attribute.

When views or relations of the global schema are created, Virtual DataPort can automatically calculate its search methods from those of the sources and the expression used to define the view. This allows the system to know \textit{a priori} whether or not a specific query can be responded to. Therefore, for derived views DataPort allows the query capacities of same to be consulted, but they cannot be modified directly.

### 6.6.6 View Configuration Properties

Virtual DataPort maintains properties for each data source and base view, which allow for specific characteristics of the underlying sources to be configured such as their distributed transaction support capacity or whether inserting operations are permitted. Section 6.5.10.1 indicated the configuration properties of the data sources. This section describes the configurable properties in each base view, depending on the type of data source they have come from.

The properties of each base view can be configured by selecting the base view from the left-hand menu, either in the “DataSources” or in the “QueryBuilder” area, and pressing the “Advanced” button so that the advanced configuration window of the base view appears. Underneath the search methods (using the scroll bar where necessary) is the “Wrapper Source Configuration” link that leads to the property configuration screen. They can also be configured from VQL (see the \textit{Advanced VQL Guide} [VQL] for more information). The configurable properties are as follows:

- **Allow Insert**: This indicates whether the data source underlying the view accepts inserting operations. It is applicable to relational databases (accessible via JDBC and ODBC) and CUSTOM wrappers. The possible values are:
  - Default: VDP assigns a default value depending on the source type. In the case of relational sources and MY wrappers, the default value is “Yes”.
  - Yes: the data source allows for inserting operations.
  - No: the data source does not allow for inserting operations.

- **Allow Delete**: This indicates whether the data source underlying the view accepts row deleting operations. It is applicable to relational databases (accessible via JDBC and ODBC) and CUSTOM wrappers. The possible values are:
  - Default: VDP assigns a default value depending on the source type. In the case of relational sources and MY wrappers, the default value is “Yes”.
  - Yes: the data source allows for deleting operations.
  - No: the data source does not allow for deleting operations.
- **Allow Update**: This indicates whether the data source underlying the view accepts row updating operations. It is applicable to relational databases (accessible via JDBC and ODBC) and CUSTOM wrappers. The possible values are:
  - Default: VDP assigns a default value depending on the source type. In the case of relational sources and MY wrappers, the default value is "Yes".
  - Yes: the data source allows for updating operations.
  - No: the data source does not allow for updating operations.
- **Delegate All Operators**: This indicates whether the view allows for all operators to be delegated. Applicable to MY wrappers. The value is "true" by default.
- **Delegate AND Condition**: This indicates whether the view allows for the AND condition to be delegated. The value is "true" by default for base views from CUSTOM wrappers.
- **Delegate Array Literal**: This indicates whether the view allows for array-type compound constants to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Compound Field Projection**: This indicates whether the view allows projections on compound fields to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Left Function**: This indicates whether the view allows for conditions with functions on the left part to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Left Literal**: This indicates whether the view allows for conditions with constants on the left part to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate NOT Condition**: This indicates whether the view allows for the NOT condition to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate OR Condition**: This indicates whether the view allows for the OR condition to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Register Literal**: This indicates whether the view allows for register-type compound constants to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Right Field**: This indicates whether the view allows for conditions with fields in the right part to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Right Function**: This indicates whether the view allows for conditions with functions on the right part to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Delegate Right Literal**: This indicates whether the source allows for conditions with constants on the right part to be delegated. Applicable to CUSTOM wrappers. The value is "true" by default.
- **Supports Distributed Transactions**: This indicates whether the underlying data source can take part in an XA distributed transaction. It is applicable to relational databases (accessible via JDBC and ODBC) and CUSTOM wrappers. The possible values are:
  - Default: VDP assigns a default value depending on the source type. In the case of relational sources and CUSTOM wrappers, the default value is "Yes".
  - Yes: the data source meets the XA specification.
  - No: the data source does not meet the XA specification.
- **Data in Order Field List**: This indicates the fields by which the data is ordered. They are separated by commas, and the field names are accompanied by the `ASC/DESC` modifier. Property applicable in all data sources.
- **Delegate Operators List**: This property determines the list of operators that can be delegated to the data source. This allows for VDP to optimize the query plan created using the query made by the user, delegating part of the processing to the native source. While VDP carries out this action automatically on relational databases (leaving selection, projection, union or join operations to be executed by the database that the specific base view comes from), other source types do not provide this information in their metadata, despite this sometimes being possible. VDP allows for the list of operators that can be delegated to be indicated in the Web Service and CUSTOM data type (all operators can be delegated by default in CUSTOM wrappers (DelegateAllOperators = true). Operator "=" can be delegated in WS wrappers.

### 6.6.7 Configuration Properties for Specific View Types

The base views from some types of sources include specific configuration properties. At present, the sources with base views that have specific parameters are the Denodo Aracne type (see section 6.5.7) and the Google Mini type (see section 6.5.6).
The specific properties of this type of base view can be configured by selecting the base view from the left-hand menu, either in the “DataSources” or in the “QueryBuilder” area, and clicking on the “Advanced” button so that the advanced configuration window of the base view appears. Underneath the search methods (using the scroll bar, where necessary) is the “Wrapper Specific Configuration” link that leads to the property configuration screen.

In the case of base views from Google Mini-type sources, the configurable properties are: “Site Collections”, “Client”, “Languages” and “Number of Key match”. See section 6.5.6 for a description of the function of these properties.

In the case of base views from Denodo Aracne-type sources, the configuration properties are those relating to the attributes created during the importing of the base view to contain the most relevant terms of a document (or a field thereof). See section 6.5.7 for a description of the function of these properties.

6.7 STORED PROCEDURES

DataPort allows writing stored procedures using the JAVA language (see Advanced VQL Guide [VQL]).

The DENODO_HOME/samples/vdp/storedProcedures path includes different example stored procedures. The README file in this path describes their function and contains the instructions to compile and install them. The stored procedure CalculateAvgRevenue, included in these examples, will be used in this section.

6.7.1 Importing Stored Procedures

To import a new stored procedure, access the “Query Builder” tab and, after selecting the project to which the new procedure is to be added, click on the ‘New’ option that appears beside the current list. This provides access to the screen where a new stored procedure is imported (see Figure 76). The following data are requested on this screen:

- Name of the stored procedure in Virtual DataPort.
- Full name of the class that implements the stored procedure. For example, supposing that the examples of stored procedures included with the distribution of DataPort have been compiled and installed, the value com.denodo.vdp.demo.storedprocedure.CalculateAvgRevenue can be entered. The class indicated must be in the DataPort CLASSPATH when the server is booted. For this, the class may be included from the path DENODO_HOME/extensions/dev/target/classes. The class may also be placed in any other path, wherever it is included in the value of the environment variable $DENODO_EXTERNAL_CLASSPATH.
- Path to the .jar file containing the class implementing the stored procedure (optional). If the class implementing the stored procedure is not in the current CLASSPATH of the DataPort server, this option can be used to provide the path to the .jar file that contains it.
On pressing "OK", the screen displaying the properties of the stored procedure that has just been imported is accessed, including its input and output parameters (see Figure 77).

The parameters of a stored procedure can be of types IN, OUT or IN_OUT. The IN parameters are the required input parameters to execute the procedure; the OUT parameters are the ones returned by the execution of the procedure; and the IN_OUT parameters are input parameters that are also returned by the procedure.

In Figure 77 information about the CALCULATEAVGREVENUE is shown; it has two parameters: the input parameter (IN) called taxid_list, of type array, and the output parameter (OUT) avgrevenue, of type double.
6.7.2 Executing Stored Procedures

Stored procedures can be invoked using the `CALL` statement (Advanced VQL Guide [VQL]).

Stored procedures can also be executed from the administration tool. To do so, press the ‘Execute’ button from the screen displaying the stored procedure properties. The tool will then request values for the procedure input parameters and will execute it, displaying the output obtained.

Any query execution can be cancelled at any moment. If results are not being shown yet, cancellation can be achieved by pressing the ‘[X]’ button of the ‘Please Wait’ dialog which can be seen until the first batch of results is shown. Once the query results are being shown, the execution can be stopped by pressing the ‘Stop’ button.

Figure 78 shows the screen in which the values of the input parameters for the procedure `CalculateAvgRevenue` are specified. This sample procedure receives a parameter of type array of registers as input called `taxid_list`. Each register contains a single field known as `taxid`, which corresponds to a client’s Tax ID. In our example, an array containing two registers with the values ‘B78596011’ and ‘B78596012’ can be created (section 6.4.4.1.4 explained how to use the creation wizard for compound-type constants).
Figure 78 Execution of the Stored Procedure `CalculateAvgRevenue`

The result of the execution is shown in Figure 79.
6.7.3 Use of Stored Procedures in Creating Views

Virtual DataPort stored procedures can be used as a basis for creating new derived views.

More specifically, an invocation of a stored procedure may appear in the `FROM` clause of any query or VQL view definition. The stored procedure will be considered a view whose schema includes all input and/or output parameters (that is, of type `IN`, `OUT`, `IN_OUT`) of the stored procedure, and with a single search method (see section 6.6.5). The search method of a view generated from a stored procedure will have one obligatoriness (OBL) constraint for every input parameter (IN or IN_OUT), and an NOS constraint for every output parameter (OUT).

A view can be graphically created that uses a stored procedure in the `FROM` clause. To do so, simply ‘drag&drop’ the required procedure on the creation screen of the required type of view (union, join, selection, projection, etc.). At that time, if the procedure includes input parameters, a value will be requested for them.
7 PUBLICATION OF VIEWS AS WEB SERVICES

Virtual DataPort allows one or several views to be published as a Web Service to enable use by any external application. Each Web Service generated will be exported in two versions: one based on SOAP (SOAP) and a REST-style version which will use http as transport protocol. This section describes this publication process.

Views are published using the ‘Data Export Tool’ tab in the administration tool. Once this tab has been accessed, the steps to generate the Web service are as follows:

1. Click on the ‘new’ option of the “Web Services” panel to access the publication screen for a new Web service (see Figure 80).

2. Select a view (or stored procedure) to export using the “Projects” drop-down menu on the left and drag&drop to the work area on the right of the screen. In our example, drag&drop the `inc_grouped_by_pref_clients` view created in section 6.4.5.

3. On adding the view or stored procedure to be published, DataPort analyzes the query capabilities permitted by it and automatically generates a series of “candidate operations” to consult the view through the Web service (Figure 81 shows the operations generated for our example). Operations with the following characteristics are normally generated:

   a. The operations generated will include a series of input parameters corresponding to attributes of the view (in the case of stored procedures, they will correspond to the input parameters of the procedure). Their invoking will return any view tuples or stored procedure with values for the
input attributes that match those specified for the corresponding parameters. There is one operation of this type in our example: \texttt{getINC\_GROUPED\_BY\_PREF\_CLIE\_001}, which receives three input parameters that correspond to attributes of the view \texttt{taxID}, \texttt{revenue} and \texttt{num\_incidences}.

b. In the event of the view including search methods without mandatory attributes (see section 6.6.5), an operation will also be generated without input parameters that, on being invoked, will return all tuples from the view. In our example, this operation is called \texttt{getINC\_GROUPED\_BY\_PREF\_CLIE}.

c. The return value of all operations generated will be an array of elements. Each of these elements will correspond to a tuple of the view and will have one subelement for each attribute.

![Figure 81 Operations created for the view inc\_grouped\_by\_pref\_clients](image)

4. It is possible to delete a candidate operation (by clicking on the ‘drop’ option alongside its name) and to change its name (by clicking on the ‘rename’ option).

5. It is also possible to delete input parameters for an operation (by clicking on the ‘drop’ option alongside the name of the parameter), although depending on the query capabilities of the view (see section 6.6.5) some operations generated may include compulsory parameters that cannot be deleted.

6. It is possible to add operations generated based on other views to the Web service generated. To do so, simply drag & drop from the new view and repeat steps 3-5.

7. It is then necessary to specify:
   a. The name required for the Web service to be generated (‘Edit Web Service name’ field)
   b. The path where the .wsdl and .war files of the Web service are to be generated.
i. The .war file contains the Web Service implementation both in its SOAP and REST version, and may be installed in a J2EE Web container (such as Apache Tomcat [TOM]).

ii. The .wsdl file [WSDL] defines the interface of the SOAP-based Web service generated. This can be used with a utility for Web Service programming (such as those included in Apache Axis [AXIS]) to generate the necessary stubs to implement a client program accessing the SOAP Web service. To understand how to access the REST version of the Web Service, please see section 7.1.

8. By using the ‘Advanced’ button, it is possible to configure two more aspects of the Web Service export stage:
   a. The style of the SOAP Web Service to generate (RPC or DOCUMENT). Some consumer applications may require a concrete style for it to work properly.
   b. The data type that must be used to export the date-type attributes in DataPort. The options offered are the XML Schema types date and dateTime. The date XML Schema only considers day, month and year, while dateTime includes complete time information that includes hours, minutes, seconds and milliseconds.

9. Finally, by clicking on ‘ok’ the .wsdl and .war files will be generated in the specified path.

Table 1 shows the conversions applied between the DataPort data types and the data types used for the input and output parameters of the Web service generated. In this table, the xsd prefix identifies the XML Schema types and the soapenc prefix the SOAP types.

<table>
<thead>
<tr>
<th>DataPort Data Type</th>
<th>SOAP Web Service Data Type</th>
<th>REST Web Service Data Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>text</td>
<td>xsd:string</td>
<td>xsd:string</td>
</tr>
<tr>
<td>long</td>
<td>xsd:long</td>
<td>xsd:long</td>
</tr>
<tr>
<td>double</td>
<td>xsd: double</td>
<td>xsd: double</td>
</tr>
<tr>
<td>int</td>
<td>xsd:int</td>
<td>xsd:int</td>
</tr>
<tr>
<td>boolean</td>
<td>xsd:boolean</td>
<td>xsd:boolean</td>
</tr>
<tr>
<td>link</td>
<td>xsd:string</td>
<td>xsd:string</td>
</tr>
<tr>
<td>float</td>
<td>xsd:float</td>
<td>xsd:float</td>
</tr>
<tr>
<td>money</td>
<td>xsd:double</td>
<td>xsd:double</td>
</tr>
<tr>
<td>time</td>
<td>xsd:int</td>
<td>xsd:int</td>
</tr>
<tr>
<td>date</td>
<td>xsd:dateTime or xsd:date</td>
<td>xsd:dateTime or xsd:date</td>
</tr>
<tr>
<td>blob</td>
<td>xsd:hexBinary</td>
<td>xsd:hexBinary</td>
</tr>
<tr>
<td>xml</td>
<td>xsd:string</td>
<td>xsd:string</td>
</tr>
<tr>
<td>enumerated</td>
<td>xsd:string</td>
<td>xsd:string</td>
</tr>
<tr>
<td>array</td>
<td>xsd:complexType</td>
<td>xsd:complexType</td>
</tr>
<tr>
<td></td>
<td>(restriction on</td>
<td>(sequence)</td>
</tr>
<tr>
<td></td>
<td>soapenc:array)</td>
<td></td>
</tr>
<tr>
<td>register</td>
<td>xsd:complexType</td>
<td>xsd:complexType</td>
</tr>
<tr>
<td></td>
<td>(sequence)</td>
<td>(sequence)</td>
</tr>
</tbody>
</table>

Table 1 Conversions between DataPort data types and Web service parameter types

7.1 INVOKING THE EXPORTED REST WEB SERVICES

This section describes how to invoke the REST version of the Web Services that have been exported by DataPort, once the generated .war file has been deployed in a J2EE web container (as, for instance, Apache Tomcat [TOM]).

The generated .war file contains a webapp with the name chosen for the Web Service at export time. In the /rest path of such webapp, an information screen can be accessed which shows the available operations.
Example: if the J2EE application server is running on port **8080** of the **acme** host, and the name chosen for the exported web service was **testWS**, the access URL for the information page would be:

```
http://acme:8080/testWS/rest
```

For each operation, the input and output parameters are shown, plus a link to the .xsd file which describes the schema of the XML document which will return the call of each operation.

To access the XML Schema of the data returned by invoking an operation, the following URL format should be used:

```
http://host:port/serviceName/rest/opName/xsd
```

Example: again, if the J2EE application server runs on port **8080** of the **acme** host, and the name chosen for the exported web service was **testWS**, the following URL will obtain the XML Schema of the data returned by the operation **getINTERNET_INC**:

```
http://acme:8080/testWS/rest/getINTERNET_INC/xsd
```

The format used to invoke a specific operation is the following:

```
http://host:port/serviceName/rest/opName?paramName1=value1&&...&paramNamn=valuen
```

, where **n** is the number of parameters of the operation.

Example: the J2EE application server runs on port **8080** of the **acme** host, and the name chosen for the exported web service was **testWS**. Let’s also suppose that the service has an operation called **getINTERNET_INC** that requires no parameters. The operation can be invoked as follows:

```
http://acme:8080/testWS/rest/getINTERNET_INC
```

If the operation to be invoked is **getINTERNET_INCBYIINCID**, which requires one input parameter called **iinc_id**, the results when this parameter has a value equals to **1** would be obtained by writing:

```
http://acme:8080/testWS/rest/getINTERNET_INCBYIINCID?iinc_id=1
```

Both operations described above can be easily generated by using the **internet_inc** view created in section **6.3.4**.

If some parameter is of compound-type, its value will be represented by using the **ROW** and **{ }** VQL constructors (see section **6.4.4.1.4** and VQL Advanced Guide [VQL]).

Example: let’s suppose that the J2EE application Server is running on port **8080** of the **acme** host, and the name chosen for the exported web service was **testWS**. Let’s also suppose that the service contains an operation called **getREVENUESUM**. This operation (which can be easily generated from the **REVENUESUM** view that was created in section **6.4.4.1.4**) requires a record array-type parameter called **clients**. Each record has just one text-type field that represents the company’s tax identifier. The operation will return the corresponding amount by adding every sale done to each company in the array. The call would be:

```
http://acme:8080/testWS/rest/getREVENUESUM?clients={ROW('B78596011'),ROW('B78596012'))}
```
7.2 CONFIGURING THE EXPORTED WEB SERVICES CONNECTIONS

The exported web services need to be deployed in a J2EE application server. Each invocation on a web service operation will execute a statement on the DataPort server to obtain the desired results.

It is possible to configure certain parameters of the connections used by the web service in order to access DataPort.

The most important aspect that can be configured is the use of a connection pool. This way, the connections used by the web service to access DataPort can be reused. In production environments, the use of the connection pool is strongly recommended.

The configuration of the connection parameters is specified in the `web.xml` file contained in the `.war` file generated by the exporting process. The parameters that can be configured are:

- `chunkSize`, `chunkTimeout`, `queryTimeout`. Their function is the same as in other VDP clients (see DataPort Development Guide [DEV]).
- `poolEnabled`. It can take the `true` or `false` values to, respectively, indicate that the pool should be used or not.
- `poolInitSize`. Initial number of connections opened by the pool.
- `poolMaxActive`. Maximum number of connections in the pool.

In the `web.xml` file, each parameter appears as an environment variable for the web service. The property name appears as the value of the variable sub-element `env-entry-name`. The value must be indicated in the `env-entry-value` sub-element. For instance, the next fragment shows the `poolEnabled` parameter configured with the `true` value.

```
<env-entry>
  <env-entry-name>poolEnabled</env-entry-name>
  <env-entry-value>true</env-entry-value>
  <env-entry-type>java.lang.String</env-entry-type>
</env-entry>
```
8 DATABASES, USERS AND ACCESS RIGHTS IN VIRTUAL DATAPORT

This section describes various key concepts in the Virtual DataPort architecture.

Section 8.1 describes the concept of databases as understood in the context of a Virtual DataPort server. Section 8.2 describes the general concepts of user and access rights management in DataPort. Finally, section 8.3 describes how to manage this structure using the Virtual DataPort graphic administration tool. The VQL commands for managing this structure are described in detail in the VQL Advanced Guide [VQL].

8.1 DATABASES IN VIRTUAL DATAPORT

A Virtual DataPort server can contain various different databases (do not confuse with the possible external databases that can act as sources of the system). A Virtual DataPort database represents a virtual schema comprised of a series of DataSources, views and base relations.

Each database is independent of the remaining databases of the server and, as described in detail in the next section, the different users can have different privileges for each database.

When a Virtual DataPort server is installed, an example database called admin is created.

8.2 USER AND ACCESS RIGHTS STRUCTURE IN VIRTUAL DATAPORT

8.2.1 User types

Denodo Virtual DataPort distinguishes 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 never 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 specific views contained therein.

8.2.2 Types of access rights

Virtual DataPort access rights are applied to a specific user to delimit what he/she can do to databases, views and stored procedures in a specific server.

Access rights for a user can be applied globally to a database or specifically to a view or stored procedure in a specific database. Access rights to particular views and stored procedures are applied only if the user does not have the corresponding access right on a global level.

Denodo Virtual DataPort supports the following types of global database access rights:

- **Read access**: If a user has this access to a database on a global level, he/she can execute the following tasks on it:
Databases, Users and Access Rights in Virtual Dataport

- View the list of base relations, views and/or stored procedures of the database (corresponds to the VQL LIST command). If a user does not have read access to a database, but does have it for some of its views and/or stored procedures, the LIST command may be executed, but it will only display the group of views and procedures to which the user has read access.
- View the schema of any base relation, view or stored procedure of the database (corresponds to the VQL DESC command).
- Execute queries on any view and/or stored procedure of the database (corresponds to the VQL SELECT command).

- **Creation access:** If a user has this access to a database on a global level, he/she can execute the following tasks on it:
  - Create DataSources, views, base relations and stored procedures in the database (corresponds to the VQL CREATE command).

- **Write access:** Having this write access automatically implies read access too. If a user has this access to a database, he/she can execute the following additional tasks on it:
  - Delete any view, base relation and/or stored procedure of the database. He/she can also delete any DataSource from databases he/she has created, but cannot delete DataSources created by other users (corresponds to the VQL DROP command).
  - Modify any view, base relation and/or stored procedure from the database. He/she can also alter any DataSource from databases he/she has created, but cannot alter DataSources created by other users (corresponds to the VQL ALTER command).

- **Connection access:** If a user has this access to a database, then he/she can connect to it, without it he/she cannot connect. This type of access is useful, for example, if you wish to temporarily revoke a user’s access to a database without having to alter the rest of his/her normal privileges manually.

Denodo Virtual DataPort also supports individual privileges to specific views and stored procedures. The types of access that can be applied to a specific view and/or stored procedure of a database are:

- **Read access:** If a user has this access to a view or stored procedure, he/she can execute the following tasks on it:
  - View its schema (corresponds to the VQL DESC command).
  - Execute queries to it (corresponds to the VQL SELECT command).
  - Create new views that use it, wherever creation access is available in the database to which it belongs. Corresponds to the VQL CREATE VIEW command.
  - If a user does not have read access to a database, but does have read access to some of its views and/or procedures, the LIST command may be executed, but only said components will be displayed.

- **Write access:** Having this write access automatically implies read access too. If a user has this access to a view and/or stored procedure, he/she may execute the following additional tasks on it:
  - Delete the component (corresponds to the VQL DROP command).
  - Alter the component (corresponds to the VQL ALTER command).

- **Insertion access:** This allows for tuples to be inserted in the view (never a stored procedure) through INSERT statements.

- **Update access:** This allows for view (never a stored procedure) tuples to be updated through UPDATE statements.

- **Deletion access:** This allows for view tuples to be deleted (never a stored procedure) through DELETE statements.
8.3 GRAPHIC ADMINISTRATION OF DATABASES, USERS AND ACCESS RIGHTS

To manage the databases, users and access rights of a Virtual DataPort server access the administration tool in server administration mode, which implies doing so with an administrator-type user (in this case, it is not mandatory to specify a database in the server connection uri).

When the server is installed, a default administrator user is created whose user name is admin and whose password is also admin.

The following sections describe how to create new databases, modify or delete them and how to create new users and, finally, how to modify or delete existing users.

8.3.1 Creating Databases

To create a new database go to the “Databases Management” tab and press the ‘new’ button.

![Creating a new database](image)

The tool will then request the following data for the new database (see Figure 82):

- Name of the database.
- Description. Optional field to include a description of the database.

8.3.2 Configuration and Deletion of Databases

To view the current list of databases of the Virtual DataPort server access the administration tool “Database Management” tab. For each database the name and description field are displayed (see Figure 83). The value of the description field may be modified by pressing the "edit" button.
There is also a series of buttons to carry out different actions on the database:

- By pressing the “Assign privileges” button, a list of all Virtual DataPort users who are not administrators is given. For each user, you can specify whether the user has global connection, creation, read and/or write access rights to the database (see Figure 84).
- The “i18n”, “Swap” and “Cache” buttons allow for the internationalization configuration to be changed (see section 5.1.8), for swapping policies (see section 5.1.6) and for the cache to be configured (see section 5.1.3), respectively, for the database. In all three cases, if the “Default Configuration” box is checked, then the default configuration will be used in the server.
- By pressing the “Export VQL” button, all database metadata can be exported to a VQL file so that it can be easily recreated in another Virtual DataPort installation. Neither the users nor the active access allocations for the database are included in the export process. The database creation statement is not included either.
Lastly, it is also possible to delete the database by pressing the “delete” button (the system requests confirmation of the operation and, due to the delicacy of the operation, requires that the user password be entered again). Note that, when a database is deleted, all the components are eliminated: DataSources, views, base relations, etc.

### 8.3.3 Creating users

To create a user go to the “User Management” tab and press the ‘New User’ button.

The tool then requests the following data for the new user (see Figure 85):

- **User login.**
- **User password.** It should be written twice so that it can be checked.
- **Description.** Optional field to include a description of the user.
- **Type of user:** Users can be of the type Administrator or Normal User (user without administration privileges).
- **Authentication type.** Users can be authenticated against DataPort (“Normal” option) or against an LDAP server registered in DataPort (see section 6.5.8 for information on how to register an LDAP server). The data to be completed is different, depending on the option chosen.
  - In the “Normal” case, a password will be requested for the user. It should be written twice so that it can be checked.
  - In the case of “LDAP”, the following data must be provided (see Figure 85):
    - LDAP server to use. Select the database where the LDAP server required has been registered using the selectable “Database”. Once this has been done, select the LDAP server using the selectable “Datasource”.

![Figure 84 Assigning privileges on a database](image_url)
- LDAP user. The name of the user in the LDAP server. For example, the value 'cn=test,ou=People,dc=denodo,dc=com' identifies the test user in an organizational unit People for the domain denodo.com.

Figure 85 Creating a normal-type user

**NOTE:** If a LDAP datasource is deleted on cascade (see section 6.3.3), then the users depending on it will also be deleted. This operation can only be executed by an administrator user.
To see the current list of users of the Virtual DataPort server go to the “User Management” tab of the administration tool. For each user the login, description field and type are shown (administrator or normal user). See Figure 87.

8.3.4 Modifying and deleting users

To see the current list of users of the Virtual DataPort server go to the “User Management” tab of the administration tool. For each user the login, description field and type are shown (administrator or normal user). See Figure 87.
The user description and password can be changed by pressing the “edit” button. If the user is not an Administrator, the option “Assign privileges” may be used to modify his/her privileges for various system databases (see section 8.3.4.2).

From the list of users screen you can also delete the user by pressing the “delete” button (the system will request confirmation of the operation). The predefined “admin” administrator cannot be deleted.

8.3.4.1 Changing active user password

The active user password can be modified using the ‘Change password’ option on the application ‘File’ menu. This option is available both in database administration mode and in server administration mode.

8.3.4.2 Modifying the privileges of a user

Privileges for different databases and views of the system can be modified for users that are not administrators.

Once a user’s privilege edit screen has been accessed (see Figure 88), a list of the VDP server databases appears. For each database a series of verification boxes appears to give the user global connection, read, create and/or write access rights for all the base relations and views of the databases.
Figure 88 Assigning privileges to a user at database level

To configure database view access rights, the option “Edit View Privileges” associated with the database should be used. Then you access a list of the views and base relations contained in the selected database (see Figure 89). For every view or stored procedure, check boxes will allow to provide users with read, write, insert, update and/or delete access (the last three for views only) to the component. It is important to note that if you have marked a check box of a certain access right on a global level in the database, then the specific rights assigned on a view level will be ignored.
Figure 89 Assigning privileges to a user at view and/or stored procedure level
9  MONITORING THE DATAPORT SERVER

It is possible to access monitoring information and events for the DataPort server using the Java Management eXtensions (JMX) standard. This information can be used, among others, to manage the use of the server and to audit the actions carried out on the data of the sources and/or the DataPort metadata.

The following is included among the monitoring information available:

- General information on the server such as the number of connections, active or otherwise, dealt with by the server or the amount of memory occupied.
- Information on the configuration and accesses to the server cache.
- Information on the accesses to each system data source.
- Information on the accesses to the views of each DataPort database.
- Information on the transactions, active or otherwise, made on the server.

Furthermore, DataPort generates different types of notifications accessible via JMX, when events such as the following occur:

- The running of a metadata creation or modification statement, commonly known as DDL statements (e.g. ALTER TABLE, CREATE VIEW, etc.).
- The running of a data query or update statement, commonly known as DML statements (e.g. SELECT, INSERT, etc.).
- The start or end of a transaction run on the server.

In JMX, the information exported is displayed as a set of objects known as MBeans. The MBeans exported by DataPort belong to the JMX domain com.denodo.vdb.management.mbeans.

Any JMX client can be used to access the DataPort monitoring information and events. JMX jConsole (included with the Java Developer Kit 1.5) and jManage consoles have been successfully tested to access the information and events exported by the DataPort server. To use these consoles, it is usually needed to specify an URL to the server. For instance, if the DataPort server is running in the port 9999 in the local host, the URL would be:

```
```

It is also possible to access the mbeans by program, using the JMX API.

Subsection 9.1 provides information on how to use jConsole to access DataPort. The following subsections describe the information and events available in each of the following categories in further detail: general server information, general data source information, general cache information, information and events on DDL statements, information and events on DML statements and information and events on transactions.

9.1  USING JCONSOLE

jConsole is a server administration console compliant with the JMX standard that is distributed with the Java Developer Kit as of version 1.5. This section provides a brief introduction to the use of jConsole for accessing DataPort monitoring information and events. Consult the Java Developer Kit documentation for detailed information on the use of jConsole.

To run jConsole, simply use the executable file included with the Java Developer Kit. For example, in Windows systems:
In the agent connection window, select the 'Remote' tab, enter the DataPort server data (machine, port, user name and password) and click on the 'Connect' button. An administrator-type user must be used.

![Connection to DataPort from jConsole](image)

On connecting to the server, different information is displayed by jConsole by default for a monitored agent such as the time that the server has been raised or information on the number of threads and the use of memory by the process.

The specific DataPort server information is in the 'MBeans' tab. This tab displays the MBeans currently available to the left of the screen. The DataPort information is provided through the MBeans of the domain com.denodo.vdb.management.mbeans.
The information and events provided by each of the MBeans available in this package are described in the following sections. This section only shows an example to illustrate the use of jConsole with DataPort.

**Example:** access to the information and events regarding active connections and DDL statements.

On the left of Figure 91 are the MBeans containing the information on DataPort. All of these are uploaded to the domain com.denodo.vdb.management.mbeans. On the right of the figure are the properties of the MBean `com.denodo.vdb.management.mbeans:type=VDBServerManagementInfo` (see section Figure 94), which include the number of active connections and the number of total connections, since the server was started. The values of the properties that can be modified are displayed by jConsole in blue (e.g. the value of the `CompleteManagementActive` property is blue).

The information and events regarding DML statements (SELECT, INSERT, UPDATE, DELETE, etc.) are accessible via the MBeans uploaded to `com.denodo.vdb.management.mbeans:type=RequestsManagementInfo` (see section 9.6). By clicking on this name, a list of the existing databases in the DataPort server will be displayed. By clicking on one of them, information on the DML statements run on this database will be accessed. For example, Figure 92 shows information on the queries run on the admin database. Information such as the total number of requests dealt with (TotalRequests property) or the number of active requests at that time (ActiveRequests property) is displayed.
Click on the 'Refresh' button to refresh the information on the attributes displayed on the current screen by querying the server. Note, however, that each DDL request on DataPort appears as a new attribute. For jConsole to display new attributes (and, therefore, data corresponding to new requests), you must access the screen of any other MBean and then return to the original screen. This will ensure jConsole loads the new attributes.

To view the data on each request, double click on its 'Value' cell and all its data will be accessed such as the type of statement to have been run (SELECT, INSERT, etc.), the user to have run it, the time at which it was run or the exact statement run (see Figure 93).
Continuing with the example, the events related to DML statements from the *admin* database can be seen by clicking on the ‘Notifications’ tab and then the ‘Subscribe’ button. From then on, every time a DML statement is run in DataPort, a new notification will be displayed in *jConsole*. Double click on its ‘UserData’ cell for details on the statement run (see Figure 94).

**Figure 93** Details of a query in *jConsole*
9.2 GENERAL INFORMATION ON THE SERVER

General information on the server is accessed through the MBeans 
com.denodo.vdb.management.mbeans:type=VDBServerManagementInfo and 
com.denodo.vdb.management.mbeans:type=MemoryManagementInfo. The properties of 
each one are described below:

- **com.denodo.vdb.management.mbeans:type=VDBServerManagementInfo.**
  - **CompleteManagementActive:** This property can be modified by the user and can take 
    the values true or false. If its value is set to true, complete monitoring mode is enabled. In this 
    mode, an estimate of the current memory (TotalMemory property of the MBean MemoryManagementInfo) and the memory peak occupied by a query (MaxMemory property of the MBean MemoryManagementInfo) are calculated. The monitoring of these 
    parameters is costly and, therefore, this mode should only be enabled when necessary.
  - **ServerName:** (read only) DataPort server URI [e.g. 
    //localhost:9999/VQIServer]
  - **TotalConnections:** (read only) Total number of connections open against the server, 
    since it was started.
  - **ActiveConnections:** (read only) Number of active connections at any given time.
Monitoring the dataport Server

- com.denodo.vdb.management.mbeans:type=MemoryManagementInfo. The attributes of this MBean are only calculated, if complete monitoring mode is enabled (see above).
  - TotalMemory. (read only) Estimate of the total memory occupied by the server.
  - MaxMemory. (read only) Estimate of the memory peak reached by the server since its start.

Furthermore, the MBean com.denodo.vdb.management.mbeans:type=MemoryManagementInfo provides an operation that can be invoked by a JMX client:
- gc. This runs the “garbage collector” of the virtual JAVA machine on the DataPort server.

9.3 GENERAL INFORMATION ON DATA SOURCES

The general information on data sources is provided via the MBeans as follows:

com.denodo.vdb.management.mbeans:type=DataSourceManagementInfo,datasourcetype=<dstype>,datasourcename=<dsname>.

Where <dstype> is the type of data source (JDBC, ODBC, WS, XML, etc.) and <dsname> is the name of the data source.

For each data source, its MBean provides the following properties:

- DatabaseName. (read only) Name of the database housing the data source.
- DataSourceName. (read only) Name of the data source.
- DataSourceType. (read only) Type of the data source: JDBC, ODBC, XML, WS (Web services), DF (delimited files), ARN (Aracne), GS (Google Mini), CUSTOM, LDAP, HTML.
- NumRequests. (read only) Number of requests made on the data source since the start of the server.
- ActiveRequests. (read only) Number of requests on the data source active at any given time.

9.4 INFORMATION ON THE CACHE

The general information on the server is accessed via the MBeans as follows:

com.denodo.vdb.management.mbeans:type=Cache,databaseName=<dbname>,subtype=DatabaseCache,

where <dbname> specifies the name of a DataPort database, and

com.denodo.vdb.management.mbeans:type=Cache,databaseName=<dbname>,subtype=ViewCache,viewName=<vname>,

where <dbname> specifies the name of a DataPort database and <vname> specifies the name of a view of said database.

Below is a description of the properties of each one of these two subtypes:

- subtype=DatabaseCache. The following properties are given for each DataPort database:
  - DatabaseName. (read only) Name of the database.
  - CacheStatus. (read only) Status of the cache (ON/OFF/POST).
  - UserName. Name of the user in the data source associated with the database used as cache.
  - UserPwd. Password of the user in the data source associated with the database cache.
- **DriverClassName**: Class of the JDBC driver in the data source associated with the database used as cache.
- **LastCacheAccessStatus**: (read only) This indicates whether the last access to the cache database was successful (`true`) or not (`false`).
- **CustomDataSource**: (read only) This indicates whether the database used as cache uses (`true`) or not (`false`) a different data source to that used by default by the DataPort server.
- **DBURI**: (read only) Connection URI to the database used as cache.

**subtype=ViewCache**: The following properties are given for each view (viewName) of every database (databaseName):

- **DatabaseName (RO)**: (read only) Name of the DataPort database to which the view belongs.
- **ViewName**: (read only) Name of the view.
- **CacheStatus**: Status of the view cache (ON/OFF/POST).
- **TTLInCache**: Lifetime of the data in the view cache.
- **LastAccess**: (read only) Moment of the last access to the view cache.
- **LastRefresh**: (read only) Moment in which new tuples were last written in the view cache.
- **RefreshFailuresCount**: (read only) Number of failures recorded in the access to write new tuples in the view cache.

### 9.5 INFORMATION AND EVENTS ON CATALOG ACCESS (DDL STATEMENTS)

The information and events on catalog information access (DDL statements) are given via MBeans as follows:

```
com.denodo.vdb.management.mbeans:type=CatalogManagementInfo,databaseName=<dbname>,
```

where `<dbname>` specifies the name of a DataPort database.

The MBeans of this type offer a single property (read only) known as `DatabaseName` that indicates the name of the database.

On subscribing to the MBean of a certain database, notification is received every time one of the following events occurs:

- A transaction starts.
- A statement affecting the metadata of the database is run.
- A transaction ends.

The following fields appear in each notification:

- **Timestamp**: Moment in which the notification of the statement is generated in the JMX server.
- **Type**: Indicates whether it refers to the start of a transaction (`'startTransaction'`), the end of a transaction (`'endTransaction'`) or the running of a catalog access statement (`'newEvent'`).
- **UserData**: compound element. Its subproperties depend on the type of notification:
  - **Statement run**:
    - **DatabaseName**: Name of the DataPort database on which the statement is run.
    - **UserName**: ID of the user running the statement.
    - **EventID**: ID of the notification generated by DataPort. The IDs are allocated using a counter.
    - **TransactionID**: ID of the transaction within which the statement is run.
    - **Time**: Moment at which the statement is run according to the DataPort server.
    - **VQLQuery**: VQL code for the statement.
  - **Transaction start**: ...
• **Autostarted**: This indicates whether the transaction has been explicitly created by the user. All statements modifying the DataPort catalog must be run within a transaction. Therefore, if the user has not created it, DataPort will create it itself.
• **DatabaseName**: Name of the DataPort database on which the transaction is started.
• **UserName**: ID of the user starting the transaction.
• **EventID**: ID of the notification generated by DataPort. The IDs are allocated using a counter.
• **TransactionID**: ID assigned to the starting transaction.
• **Time**: Moment at which the transaction starts according to the DataPort server.

**Transaction end:**
• **DatabaseName**: Name of the DataPort database on which the transaction is run.
• **UserName**: ID of the user running the transaction.
• **Status**: Completion status of the transaction (**COMMIT** or **ROLLBACK**).
• **EventID**: ID of the notification generated by DataPort. The IDs are allocated using a counter.
• **TransactionID**: ID assigned to the ending transaction.
• **Time**: Moment at which the transaction starts according to the DataPort server.

**SeqNum**: Sequence number generated by DataPort for the notification. Normally, the sequence numbers are assigned using a counter.

**Message**: 
• If the running of a statement is notified: Processed the `<event eventId>` in transaction `<transactionId>`.
• If the start of a transaction is notified: Started the transaction `<transactionId>`.
• If the end of a transaction is notified: Finished the transaction `<transactionId>`.

**Event**: This will be `javax.management.Notification[[com.denodo.vdb.management.mbeans:type=CatalogManagementInfo,databaseName=<dbName>,type=<eventType>][message=<eventmessage>]]`, where `<dbName>` is the name of the DataPort database, `<eventType>` is `startTransaction, endTransaction` or `newEvent` and `<eventmessage>` takes the same value as in the Message property.

**Source**: MBean name.

### 9.6 INFORMATION AND EVENTS ON THE RUNNING OF DML STATEMENTS

The information and events on the running of DML statements are given via MBeans as follows:

```java
com.denodo.vdb.management.mbeans:type=RequestsManagementInfo,userName=<dbname>,
```

where `<dbname>` specifies the name of a DataPort database.

The following properties are given for each database:

• **DatabaseName**: (read only) Name of the DataPort database.
• **MaxRequests**: Maximum number of requests exported at the same time in the MBean. Each request appears as an attribute in the form `Request<i>`.
• **TotalRequests**: (read only) Total number of requests recorded since the start of the server.
• **ActiveRequests**: (read only) Number of active requests (not finished).
Virtual DataPort 4.1 Administrator Guide

Monitoring the dataport Server

- **Request**: (read only) For each request run, a new compound-type attribute will appear with the following subproperties:
  - **Identifier**: transaction identifier. The identifiers are normally allocated using a counter.
  - **DatabaseName**: Name of the DataPort database on which the statement is run.
  - **UserName**: ID of the user running the statement.
  - **Cache**: This takes the value `true`, if the query has accessed the cache, while it is running, and the value `false`, where this is not the case.
  - **Memory**: Estimate of the memory consumed by the statement. Only available if the property `CompleteManagementActive` of the MBean `com.denodo.vdb.management.mbeans:type=VDBServerManagementInfo` is enabled.
  - **MaxMemory**: Estimate of the memory peak consumed by the query. Only available if the `CompleteManagementActive` of the MBean `com.denodo.vdb.management.mbeans:type=VDBServerManagementInfo` is enabled.
  - **StartTime**: Moment at which the statement started to run.
  - **EndTime**: Moment at which the statement finished its run.
  - **RequestType**: This indicates the type of statement. It can take the following values: `SELECT BASE VIEW, SELECT VIEW, QUERY WRAPPER, CALL STOREDPROCEDURE, INSERT, INSERT, UPDATE`.
  - **State**: It can take the following values: `OK, ERROR` or `PROCESSING`.
  - **Swap**: This takes the value `true`, if the server has needed to swap the intermediate results to disk, while the query is running, and the value `false`, where this is not the case.
  - **VQLQuery**: VQL code for the statement.

Furthermore, the mbeans of this type also allow for notifications to be subscribed to. Every time a DML statement is run on the specified database, a notification with the following data is received:

- **Timestamp**: Moment in which the notification of the statement is generated in the JMX server.
- **Type**: This type of notification includes the string `endRequest` as the type indicator.
- **UserData**: compound element. Its subproperties are the same as those of the `Request` property described above.
- **SeqNum**: Sequence number generated by DataPort for the notification. Normally, the sequence numbers are assigned using a counter.
- **Message**: In this type of notification, it takes the value `Finished the request`.
- **Event**: This will be `javax.management.Notification[source=com.denodo.vdb.management.mbeans:type=RequestsManagementInfo,databaseName=<dbname>,[type=endRequest][message=Finished the request '<i>'], where <dbName>` is the name of the DataPort database and `<i>` is the statement identifier.
- **Source**: MBean name.

### 9.7 INFORMATION AND EVENTS ON TRANSACTIONS

The information and events on the running of transactions are given via MBens as follows:

```
com.denodo.vdb.management.mbeans:type=TransactionsManagementInfo,databaseName=<dbname>,
```

where `<dbname>` specifies the name of a DataPort database.

The following properties are given for each database:
Monitoring the dataport Server

- **DatabaseName**: (read only) Name of the DataPort database.
- **MaxTransactions**: Maximum number of requests exported at the same time in the MBean. Each transaction appears as an attribute in the form `Transaction<i>`.
- **TotalTransactions**: (read only) Total number of transactions recorded since the start of the server.
- **ActiveTransactions**: (read only) Number of active requests (not finished).
- **Transaction <i>**: (read only) For each transaction run, a new compound-type attribute will appear with the following subproperties:
  - **Identifier**: transaction identifier. The identifiers are normally allocated using a counter.
  - **DatabaseName**: Name of the DataPort database on which the transaction is run.
  - **UserName**: ID of the user running the transaction.
  - **Autostarted**: This indicates whether the transaction has been explicitly created by the user (true). All statements modifying the DataPort catalog must be run within a transaction. Therefore, if the user has not created it, DataPort will create it itself and, in this case, the value of this property will be false.
  - **StartTime**: Moment at which the transaction started.
  - **EndTime**: Moment at which the transaction ended.
  - **State**: This indicates the status with which the transaction ended. It can take the following values: ROLLBACK or COMMIT.

Furthermore, the MBeans of this type also allow for notifications to be subscribed to. Every time a transaction starts or ends on the specified database, a notification with the following data is received:

- **Timestamp**: Moment in which the notification is generated in the JMX server.
- **Type**: This type of notification takes the string 'startTransaction' or 'endTransaction' as the type indicator, depending on whether the notification indicates the start or the end of a transaction.
- **UserData**: compound element. Its subproperties are the same as those of the `Transaction<i>` property described above.
- **SeqNum**: Sequence number generated by DataPort for the notification. Normally, the sequence numbers are assigned using a counter.
- **Message**: If the notification indicates the start of a transaction, it takes the value 'Started the transaction'. If it indicates the end of a transaction, it takes the value 'Finished the transaction'.
- **Event**: This will be `javax.management.Notification[source=com.denodo.vdb.management.mbeans:type=TransactionsManagementInfo,dbName=<dbname>][type=<type>][message=<msg>]`, where `<dbName>` is the name of the DataPort database, `<type>` is the type of notification ('startTransaction' or 'endTransaction') and `<msg>` is the notification message ('Started the transaction' or 'Finished the transaction').
- **Source**: MBean name.
10 CLUSTER ARCHITECTURES/SERVER BACKUP

DataPort can operate within a cluster architecture to provide tolerance for failures and load balancing between servers.

Typically, clustering architecture for DataPort operates as follows:

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

DataPort has been validated using the following clustering solutions:

- Linux Virtual Server architecture [LVS], using Keepalived [KALV] to check whether a server is responding at any given time.
- Windows 2000 Network Load Balancing Tool [WNLB]. This is the clustering solution included in the Windows 2000 server family.

DataPort provides different facilities to make cluster architecture configuration easier:

- DataPort provides a 'PING' script to check the status of a DataPort server. Although solutions such as Keepalived provide their own default implementations to 'check the health' of an application, the PING script provided by DataPort means that server failure situations that would not be detected by default checks are detected.
- DataPort provides a utility to easily and automatically replicate metadata and the configuration of a DataPort server in a pool of other DataPort servers. This utility can also be used to program backup copies of the metadata for a DataPort server.

The following sections describe how to install the help utilities for clustering environment configuration, how to use the PING script and how to use the utility to replicate the metadata for a DataPort server in a group of other servers.

10.1 UTILITY INSTALLATION

The utilities are in the path %DENODO_HOME%\tools\db. They are compressed in a file known as denodo-db-tools-4.1.0 that is available in .zip and in .tar.gz format.

10.2 USE OF THE PING SCRIPT

The ping script is available in the bin directory of the utility distribution. It is provided in two versions: ping.sh (for Linux systems) and ping.bat (for Windows systems).

The format in which the script is invoked is as follows:

`ping [-t timeout] [-v] [//]host[:port],`

where host is the machine where the DataPort server whose status is to be checked is housed and port is the port on which it is run (port 9999 is presumed, if this is not specified), timeout is the maximum time (in milliseconds) to wait for the server response. Option –v shows the status, via the standard output, of the server and the time taken to receive a response from it.
The ping script returns value 0, if the status check is successful, and a value different to zero, where this is not the case.

An example of running the ping command is shown below:

```
ping -t 5000 -v //localhost:9999/
```

This command checks the status of the DataPort server being run in the local machine on port 9999. A maximum of 5 seconds are waited for the server’s response and the result of the check is displayed by the standard output.

### 10.3 USE OF THE IMPORT/EXPORT SCRIPTS FOR BACKUP AND/OR REPLICATION

The import and export scripts are available in the bin directory of the utility distribution. They are provided in two versions: `import.sh` and `export.sh` (for Linux systems) and `import.bat` and `export.bat` (for Windows systems).

The export script allows for metadata from a DataPort server to be exported to a file. It is possible to export all metadata and the server configuration or only the metadata and the configuration of a specified DataPort database. The metadata exported is the same as that used in the equivalent options of the administration tool (see sections 5.3 and 8.3.2).

The format in which the script is invoked is as follows:

```
export -l login -p password -u host[:port]/[database] [-i] -f outputFilename [-P name=value]*
```

where:
- `-l login` indicates the login name with which the server connection is made. The user must have sufficient access rights for the action to be run.
- `-p password` indicates the password with which the server connection is made.
- `-u uri` Connection URI to the DataPort server. If no database is specified in the URI, all server metadata will be exported. Otherwise, only that database metadata will be exported. If no port is specified, port 9999 is presumed.
- `-i` This option exports only ITPilot-type wrappers from a database. This is a useful option to export content from an ITPilot server, or to export ITPilot wrappers from a DataPort server database so that they can be later imported to an ITPilot server. This option can only be used if the URI specifies a database. Please see [ITPILOT](#) for more information about Denodo ITPilot.
- `-f outputFilename` indicates the name of the file to which the metadata is to be exported.
- `-P name=value`. This option allows the specification of a set of properties, by following the `field equals value` format. Currently, the only property supported is `includescanners`. If this property has the value ‘yes’, the creation statements of the ITPilot wrappers that are included in the export stage will also contain the binary files corresponding to the ITPilot scanners used by the wrapper (please see ITPilot documentation [ITPILOT](#) for more information about scanners and ITPilot wrappers).

Below is an example of running the export command:

```
export -l admin -p admin -u //localhost:9999/admin -f serverexport.vql
```

This command exports the full metadata of the DataPort server being run in the local machine on port 9999. Access to the server is provided using the login `admin` with the password `admin`. The result of the export is saved to a file known as `serverexport.vql`. 
The import script allows for the importing of metadata contained in a file obtained using the export utility to a list of DataPort servers identified by their URI. The list of servers can be provided by command line or can be included in a file. Thus, it is possible to restore a backup copy made with export and to replicate the metadata of a DataPort server to a cluster list of servers.

The format in which the script is invoked is as follows:

```
import -f path-to-file [-s servers-file | -l server (-l server)*],
```

where:

- `-f path-to-file` is the file containing the metadata to be imported.
- `-l server` specifies the URI of a server to which the metadata is to be imported. The URI must be 
  `[/]host[::port]/database?user@password`, where `host` and `port` identify the machine and the port where the server is run (the port is 9999 by default), `database` specifies the database against which the authentication is to be made, `user` specifies the name of the user with which server connection is made and `password` indicates the password used.
- `-s servers-file` specifies the name of a file containing the URIs of the servers to which the metadata is to be imported. Each URI must appear on a file line in the same format used for the URIs of parameter `-l`.

For example:

```
import -f export.vql -l localhost/admin?admin@admin1 -l host2:9099/admin?admin@admin2
```

This imports the metadata contained in `export.vql` to the servers running in the local machine on port 9999 and in the machine known as `host2` on port 9099. In both cases, authentication is made against the database `admin` using the login of the same name. The password for the first server is `admin1` and `admin2` for the second.

```
import -f export.vql -s servers.conf
```

This imports the metadata contained in `export.vql` to the servers with URIs specified in the `servers.conf` file.
APPENDICES

JDBC DRIVERS

The following table shows the JDBC drivers included with DataPort 4.1. For each driver the table indicates the databases it has been tested with (together with DataPort), the class name that must be specified when importing a data source using the driver, and the URI format used by the driver.

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Class</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>Oracle 8i</td>
<td>oracle.jdbc.OracleDriver</td>
<td>jdbc:oracle:&lt;&lt;protocol&gt;&gt;:@&lt;&lt;hostName&gt;&gt;:&lt;&lt;port&gt;&gt;:&lt;&lt;databaseName&gt;&gt;</td>
</tr>
<tr>
<td>Oracle 9i</td>
<td></td>
<td>Protocols: thin (recommended), oci, oci8, kprb</td>
</tr>
<tr>
<td>Oracle 10g</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostgreSQL 7.2.3</td>
<td>org.postgresql.Driver</td>
<td>jdbc:postgresql://&lt;&lt;hostName&gt;&gt;:&lt;&lt;port&gt;&gt;/&lt;&lt;databaseName&gt;&gt;</td>
</tr>
<tr>
<td>PostgreSQL 7.4.6</td>
<td></td>
<td></td>
</tr>
<tr>
<td>PostgreSQL 8</td>
<td></td>
<td></td>
</tr>
<tr>
<td>SQL Server 8.00.194</td>
<td>net.sourceforge.jtds.jdbc.Driver</td>
<td>jdbc:jtds:sqlserver://&lt;&lt;hostName&gt;&gt;:&lt;&lt;port&gt;&gt;/&lt;&lt;databaseName&gt;&gt;</td>
</tr>
<tr>
<td>Sybase Adaptive Server Enterprise 12.5B</td>
<td>net.sourceforge.jtds.jdbc.Driver</td>
<td>jdbc:jtds: sybase://&lt;&lt;hostName&gt;&gt; :&lt;&lt;port&gt;&gt;/&lt;&lt;databaseName&gt;&gt;</td>
</tr>
</tbody>
</table>

Table 2  JDBC Drivers

The IBM DB2 and MySQL drivers and the drivers created by their manufacturers to Microsoft SQLServer and Sybase are not included with DataPort although they can be downloaded from the websites of those companies. These drivers have also been successfully tested with DataPort. The data about them are shown in the table below:

<table>
<thead>
<tr>
<th>DBMS</th>
<th>Class</th>
<th>URI</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL 4.0.15</td>
<td>com.mysql.jdbc.Driver</td>
<td>jdbc:mysql://&lt;&lt;hostName&gt;&gt;:&lt;&lt;port&gt;&gt;/&lt;&lt;databaseName&gt;&gt;</td>
</tr>
<tr>
<td>MySQL 4.1.1</td>
<td></td>
<td></td>
</tr>
<tr>
<td>MySQL 5.x</td>
<td></td>
<td></td>
</tr>
<tr>
<td>DB2 8.2</td>
<td>com.ibm.db2.jcc.DB2Driver</td>
<td>jdbc:db2://&lt;&lt;hostName&gt;&gt;:&lt;&lt;port&gt;&gt;/&lt;&lt;databaseName&gt;&gt;</td>
</tr>
<tr>
<td>SQL Server 8.00.194</td>
<td>com.microsoft.jdbc.sqlserver.SQLServerDriver</td>
<td>jdbc: microsoft:sqlserver://&lt;&lt;hostName&gt;&gt;;DatabaseName=&lt;databaseName&gt;&gt;</td>
</tr>
</tbody>
</table>

Table 3  MySQL, IBM, Microsoft and Sybase Drivers

Every other JDBC driver should work correctly with DataPort but the ones listed in the former tables have been successfully tested by importing a JDBC datasource.

As it has already been commented in section 5.1.3, when using cache in Virtual DataPort, one must configure the Database Manager to use. The DBMS versions successfully tested are: Oracle: 8i, 9i, 10g and MySQL 4.1, 5.x (which driver is not included in the DataPort distribution).

If you wish to use a JDBC driver not included with DataPort, you can use the following methods:

- You can install the new driver either in a path included in the value of the environment variable $DENODO_EXTERNAL_CLASSPATH or in the path
DENODO_HOME/extensions/dev/target/classes (where DENODO_HOME es is the installation root directory).

- Or alternatively, when creating a new JDBC data source, you can directly specify the .jar file containing the desired driver (see sección 6.3.1).
REFERENCES

[SOAP] SOAP. http://www.w3.org/TR/soap/
[WSDL] Web Services Description Language (WSDL). http://www.w3.org/TR/wsdl

References