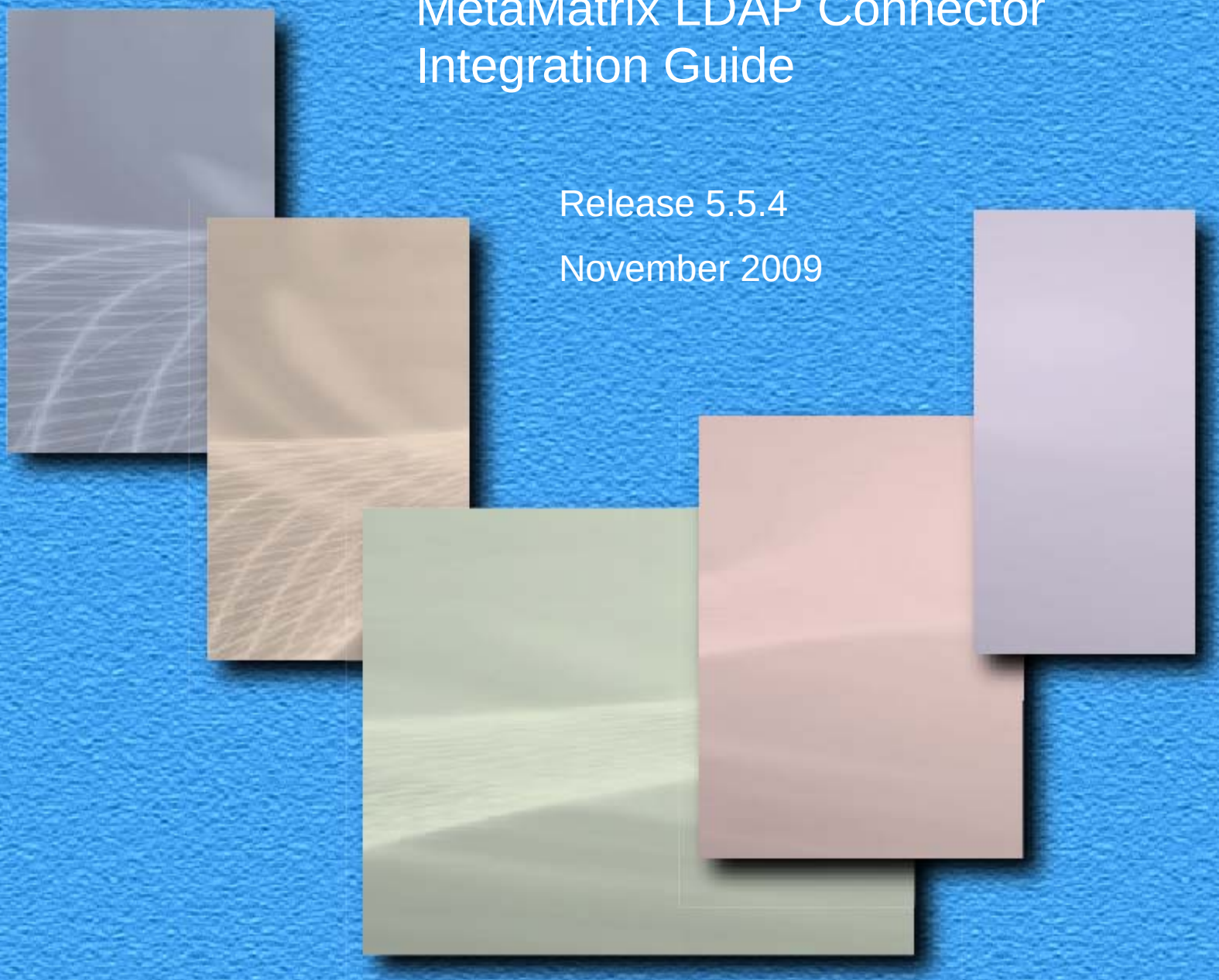


MetaMatrix LDAP Connector Integration Guide

Release 5.5.4

November 2009

© 2009 Varsity Gateway, LLC

MetaMatrix LDAP Connector Integration Guide
MetaMatrix Products, Release 5.5.4
Document Edition 1, November 2009

© 2009 Varsity Gateway LLC. All Rights Reserved.

You can obtain additional copies of this document by contacting Red Hat, Inc.

The processes and routines contained in this document are proprietary properties and trade secrets of Red Hat, Inc. Except as provided by license agreement, this document cannot be duplicated, used, or disclosed for any purpose or reason, in whole or in part, without the expressed written consent of Red Hat, Inc. The information within this document is subject to change without notice and should not be construed as a commitment by Red Hat, Inc.

MetaMatrix, the MetaMatrix logo, MetaMatrix Designer, & MetaMatrix Repository are trademarks or registered trademarks of Red Hat, Inc. in the U.S. and other countries.

Table of Contents

Chapter 1: Overview

THE LDAP CONNECTOR

The LDAP Connector

The Lightweight Directory Access Protocol, or LDAP, is a network protocol for accessing directory services over TCP/IP. The MetaMatrix LDAP Connector provides access to LDAP. This document shows how to integrate the LDAP connector into your MetaMatrix System using the MetaMatrix Tools.

THE CONNECTOR INTEGRATION GUIDE

Purpose of this Guide

This document shows how to integrate the LDAP Connector into MetaMatrix System using the MetaMatrix Designer and MetaMatrix Console tools.

This document contains an overview of using the MetaMatrix Designer to create a metadata model for the information you want to access in the LDAP instance. Information is also provided showing how to create an LDAP connector in Designer and test it. For more information, see "Design-Time Modeling and Depolyment."

For further information about the MetaMatrix tools discussed, see the appropriate MetaMatrix user's guide.

Prerequisites

This guide assumes that you have a basic knowledge of the MetaMatrix System and its component tools. This does not introduce core concepts. Rather, it provides specific information about the MetaMatrix System as it relates to the LDAP Connector. The following table lists some key concepts with which you should be familiar and where you can look to find out more information:

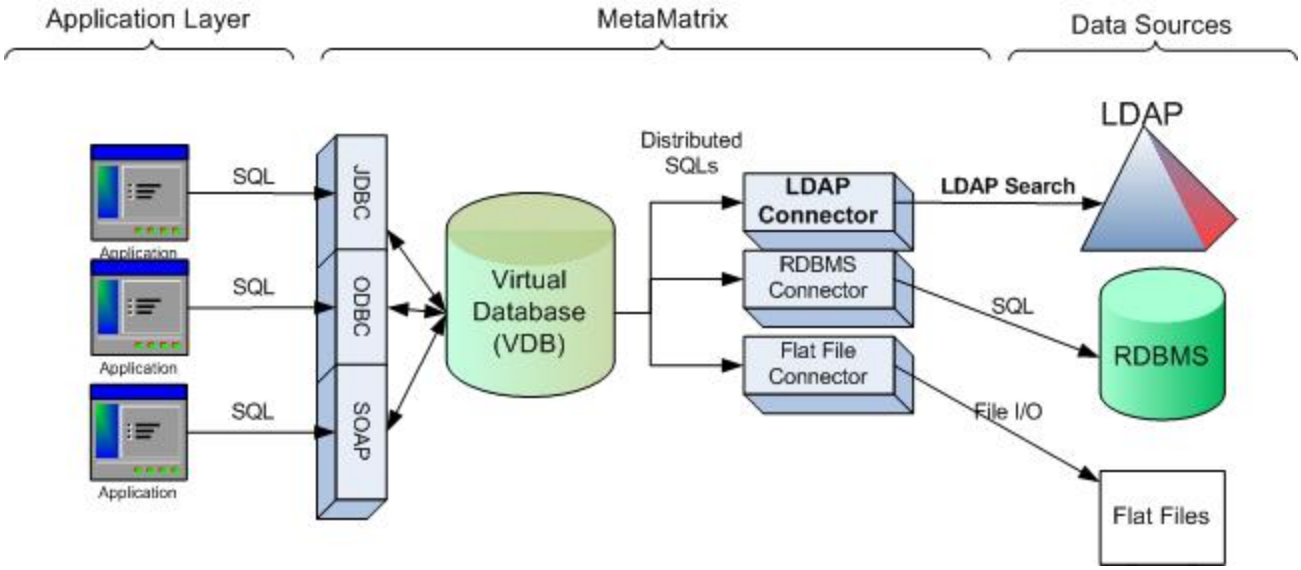
Concept	Guide
Metadata models	*MetaMatrix Designer's Guide*
Source models	*MetaMatrix Designer's Guide*
View models	*MetaMatrix Designer's Guide*
Metamodels	*MetaMatrix Designer's Guide*
Packages	*MetaMatrix Designer's Guide*
Classes	*MetaMatrix Designer's Guide*
Attributes	*MetaMatrix Designer's Guide*
Connector configuration files	*MetaMatrix Connector Developer's Guide*

Concept	Guide
Connector properties	*MetaMatrix Connector Developer's Guide*
Connector bindings	*MetaMatrix Console User's Guide*
Virtual databases	*MetaMatrix Console User's Guide*
Product Service Configurations	*MetaMatrix Console User's Guide*

Chapter 2: LDAP Connector Overview

ARCHITECTURE

The diagram below demonstrates how the LDAP Connector fits into the MetaMatrix solution architecture.

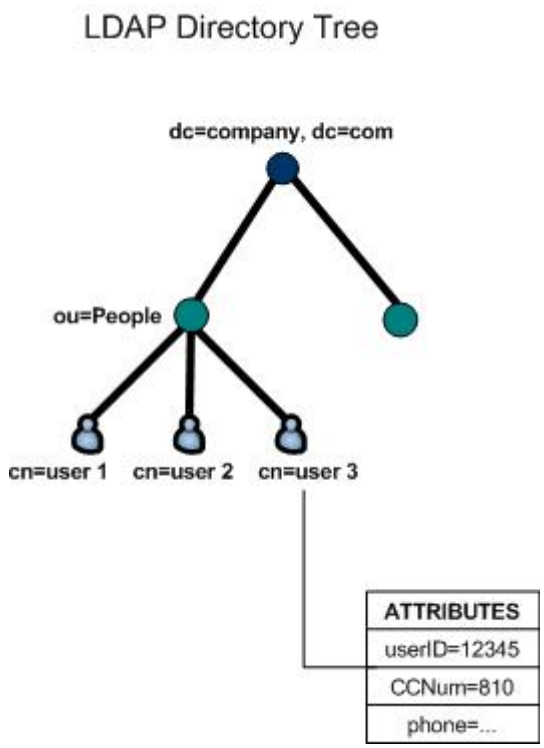


Applications submit SQL queries against a virtual database in EII via JDBC, ODBC, or SOAP. These queries are broken up into sub-queries, and sent to the corresponding data source connectors. In cases where queries are run against LDAP-based data sources, the LDAP connector receives the SQL query, and translates it into the equivalent LDAP search. The LDAP connector retrieves the data, packages it into result set batches, and returns it to MetaMatrix. MetaMatrix combines the result with any other dependent query results, performs any additional formatting or translation that was not supported by the connector, and returns results to the client application.

LDAP AND LDAP SEARCH PRIMER

The Lightweight Directory Access Protocol, or LDAP, is a network protocol for accessing directory services over TCP/IP. A directory contains a collection of related data, organized hierarchically in a tree format. Each node in the tree is a directory entry, and each entry consists of a set of attribute-value pairs. Each directory entry has a unique identifier, known as its Distinguished Name (DN). The DN consists of a Relative Distinguished Name (RDN), constructed from an attribute from the entry itself, followed by the parent entry's DN.

In the diagram below, the LDAP directory tree maintains information about people at "company". One node in the diagram is being used to contain a subtree of user entries. This node is designated an Organizational Unit, or OU, and has an RDN of ou=People. The full DN of the node is ou=People,dc=company,dc=com.



Under this OU, a number of entries exist to represent individual user's information. These leaf nodes are identified by the Common Name, or CN. For example, the third entry has an RDN of cn=user 3. This node has other attribute-value pairs associated with it that describe the person at the company. Unlike the DN, these attributes are not guaranteed to be unique.

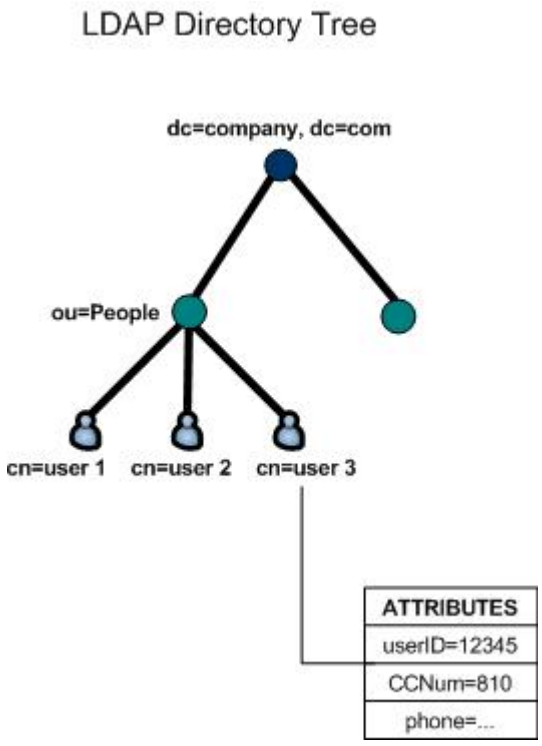
REPRESENTING LDAP DATA USING THE RELATIONAL METAMODEL

The LDAP connector, accompanied by the MetaMatrix product suite, provides a way to represent LDAP data using the Relational Metamodel. The following list describes the LDAP construct, along with the equivalent representation in a Relational model.

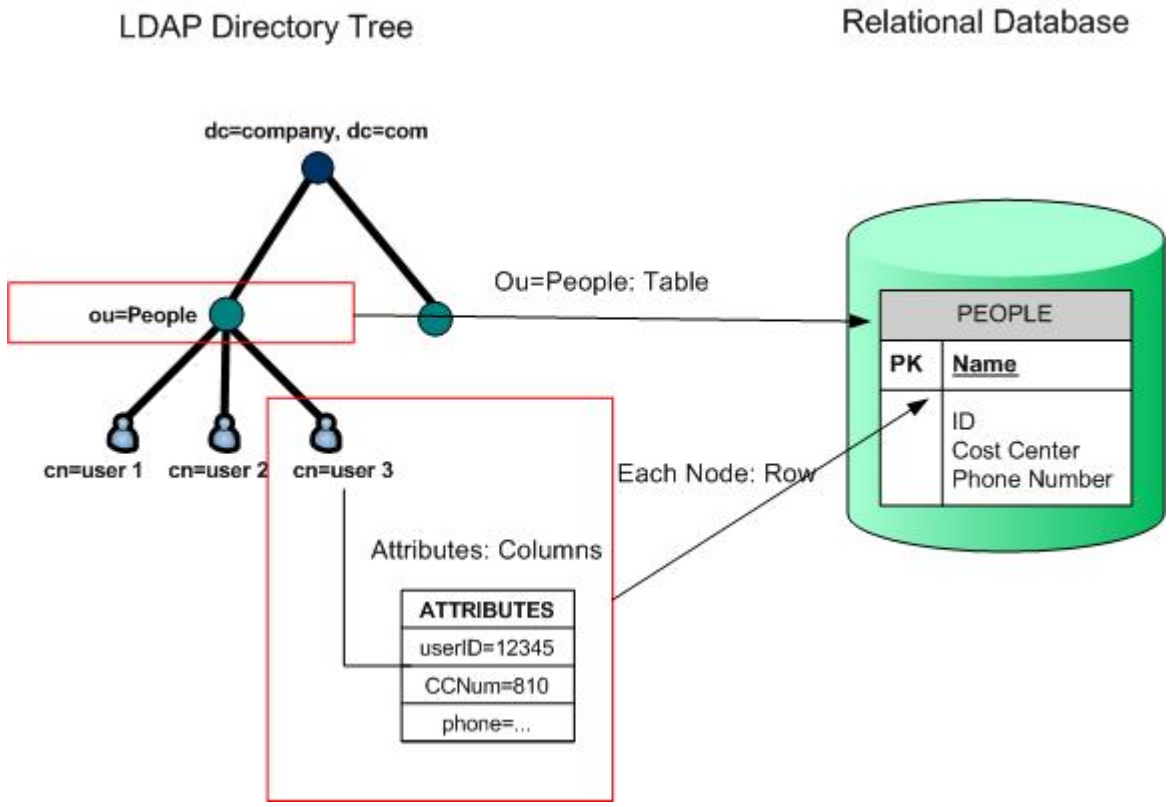
- LDAP subtrees are represented as if they were tables in a relational database.
- Each node in the subtree is represented as a row in the table.
- Each attribute of the given node can be represented as a column in the table.
- The RDN (or DN) can be used to represent a primary key[1].

For example, take the LDAP directory tree below, which maintains information about people at "company". The ou=People subtree has been selected to be represented in a Relational model.

[1] The RDN is unique within a given level of the subtree, but is not guaranteed unique if the subtree has multiple levels. In this case, the DN can be used to identify each node uniquely.



The diagram below demonstrates how the directory tree can be represented as if it were relational data.

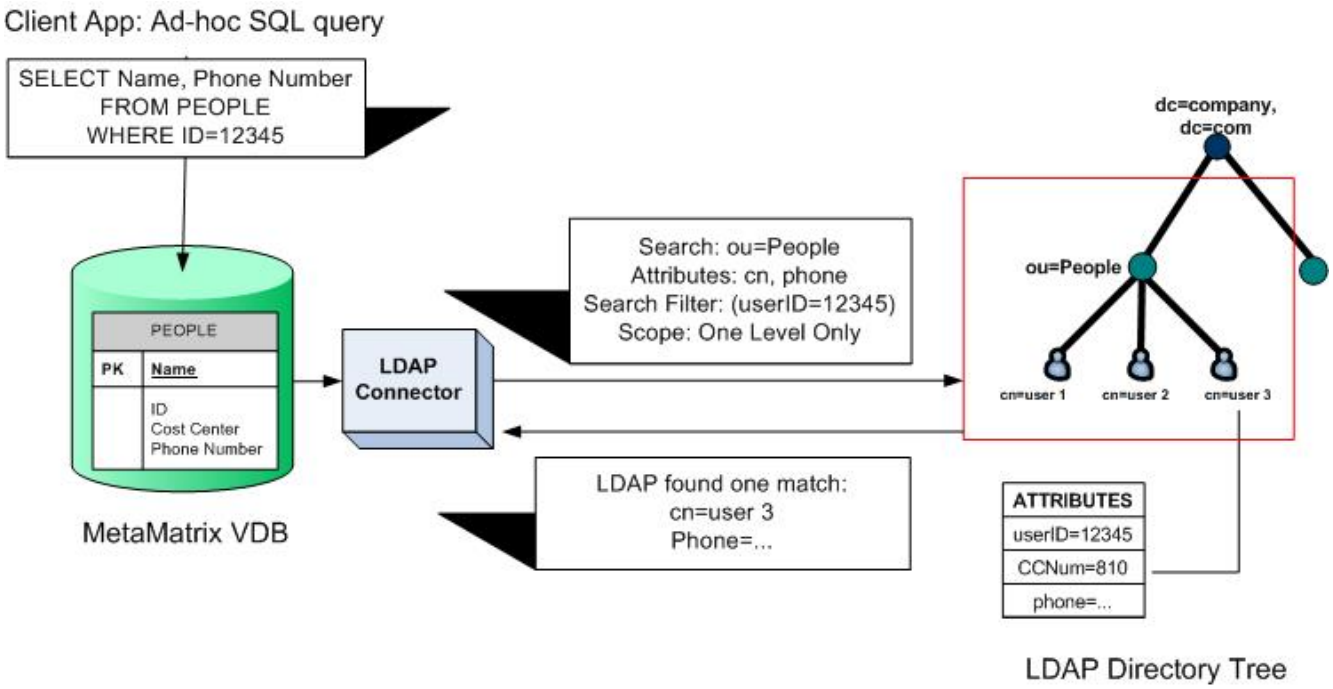


A relational model is created, and the model contains a table named "People". The table corresponds to the ou=People subtree. Each row in the table corresponds to a node in the subtree. The columns in the table correspond to attributes of each node. For example, the "userID" attribute is represented by the "ID" column name. Note that the names of the columns do not need to match the attribute names – the actual attribute name is represented by other metadata within the model (described further in the Modeling section of this document). The RDN of each node can be represented as a column, and designated as a primary key.

By default, the table represents all nodes directly under the top-level node. If the directory tree contained non-leaf nodes (i.e. it contains other subtrees), the nodes below the first level will be ignored. In LDAP terms, the search scope is restricted to "one level only". The entire subtree can be represented if a non-default search scope is specified when creating the table. The details of specifying a non-default search scope are explained in the Modeling section of this document.

TRANSLATING SQL QUERIES TO LDAP SEARCHES

The LDAP connector uses the Relational model's metadata to determine how to translate ad-hoc SQL queries into the appropriate LDAP search. The diagram below illustrates the process flow.



The process begins when a client application submits an ad-hoc SQL query to a MetaMatrix VDB (step 1). MetaMatrix determines that some part of the query uses the People table, which is bound to the LDAP connector, so it passes the subquery to the LDAP Connector. The LDAP connector converts the query into an equivalent LDAP search (step 2). The LDAP connector submits the LDAP search against the directory tree, and receives results from the LDAP server (step 3). The LDAP connector parses the results, formats a result set, and returns it to the client application.

Chapter 3: Design-Time Modeling and Depolyment

INFORMATION GATHERING

The first step in using the LDAP connector is to gather information about the LDAP data source. The Model developer should gather the following information for each LDAP subtree:

- Base OU (Table)
- Attribute Names (Columns)
- Search Scope

Additionally, Model Developers or Administrators will need to gather the following information before a Relational model containing LDAP metadata can be bound to an LDAP data source:

- LDAP Server URL (e.g. ldap://ldap.mycompany.com:389)
- LDAP Username / Password
- Security considerations (e.g. is SSL encryption required by the LDAP server for searches)

DESIGN-TIME MODELING

Overview

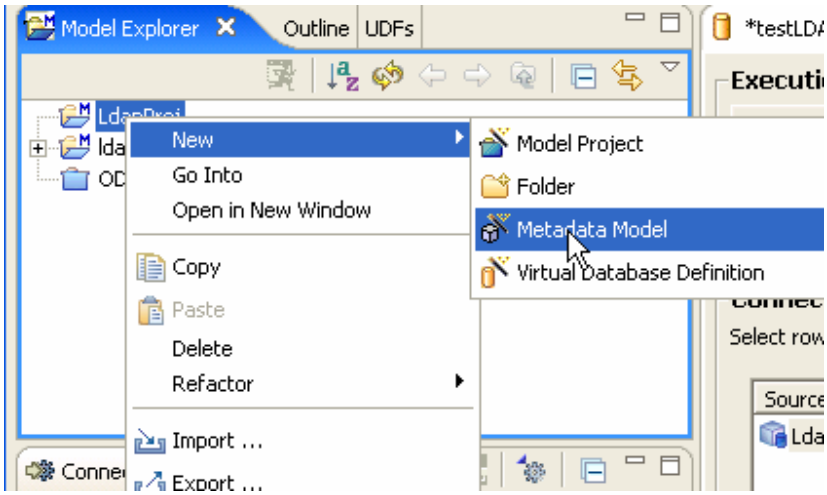
LDAP metadata can be modeled in MetaMatrix Designer using the Relational metamodel type. Each table in a given relational model represents a Base DN in LDAP. Each row in the table represents an element found in that Base DN. Each column of the table represents an attribute of the element that may exist.

In general, each table and column defines the LDAP-specific information in a special property called "Name In Source". This allows the connector to identify the attribute or Base DN name within the data source – namely, within LDAP. The actual name of the table and column can differ from the name in source if the user chooses to name them differently. This allows Designers to hide the underlying LDAP-specific names from end users if desired.

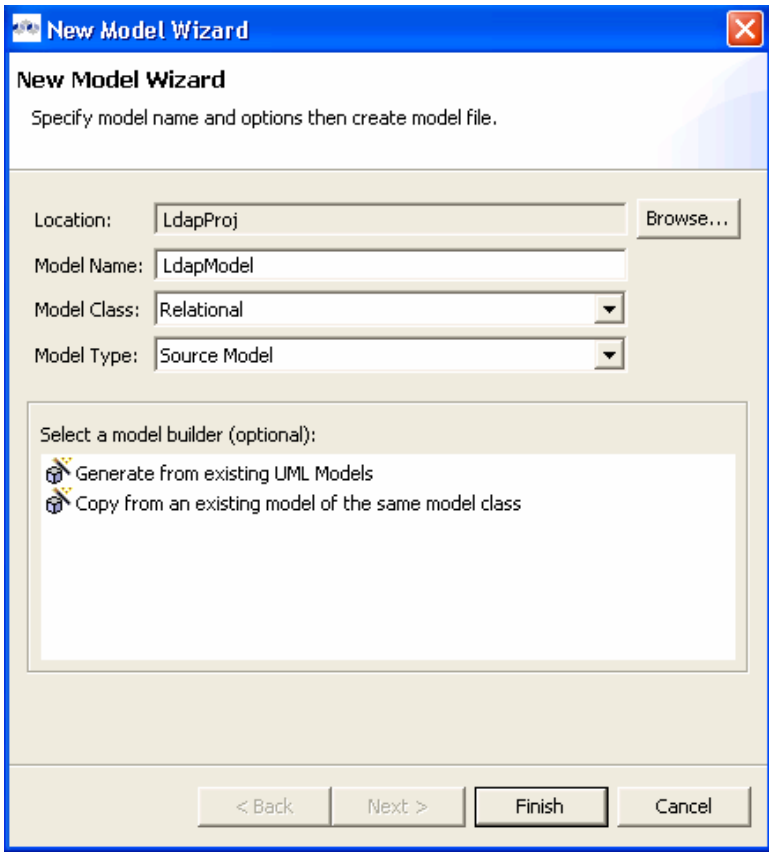
Modeling LDAP Metadata – Step-by-Step

The following is a stop-by-step example of setting up the relational metamodel in MetaMatrix Enterprise Designer for an LDAP datasource.

To begin modeling LDAP metadata, you must create a new Relational model. In MetaMatrix Designer, switch to Designer view, right-click on the Model Explorer window, and select New > Metadata Model. The New Model Wizard appears.



In the New Model Wizard, provide a name in the Model Name field. Select "Relational" for the Model Class pull-down. This allows us to represent the LDAP metadata as relational data that can be queried via SQL. Select "Source Model" for the Model Type pull-down. This indicates that the model directly describes a particular data source. Click Finish to create the new model. You are now ready to model LDAP data as a relational data source.



Select the model in Model Explorer. The properties window will display the model's properties. Set the following properties:

Supports "Distinct" = false

Supports "Join" = false

Supports "Outer Join" = false

NOTE: These properties are for informational purposes only, and do not affect the query plans or the connector behavior. They are not strictly necessary.

Select the model in Model Explorer, then right-click and select New Child > Base Table from the context menu to create a new table. Each table will correspond to data contained in a particular base DN in LDAP.

Select the table in Model Explorer. The properties window will display the table's properties. Set the following properties:

Property	Value
Name	Give the table a user-friendly name, in accordance with any established naming conventions. The name does not need to correspond to the Name In Source name.
Name in Source	Enter the base DN you wish to search when the table is queried. For example, if you wish to search ou=people,dc=company,dc=com, then enter that string (with no spaces) into the Name In Source field. Additional search info can be included using the Name In Source format as follows: search_base?search_scope?objectClass_name If you wish to specify a search scope other than the default (SUBTREE_SCOPE), you may append the JNDI-supported search scope string[2] to the base DN, using a "?" symbol to delimit the two fields. For example: ou=people,dc=company,dc=com?SUBTREE_SCOPE In addition, if you wish to restrict the search to certain objectClasses, you can add the objectClass restriction to limit the search further. For example: ou=people,dc= company,dc=com?SUBTREE_SCOPE?inetOrgPerson Finally, either of the first 2 parameters may be omitted. The following Name In Source specification will use the default search_base and search_scope and then limit to the specified objectClass: ? ? inetOrgPerson If you wish to use the default search scope, do not add any suffix.

[2] The currently supported JNDI search scope terms are ONELEVEL_SCOPE, SUBTREE_SCOPE, and OBJECT_SCOPE. Details of each option can be found in the Java SDK 5.0 documentation (http://java.sun.com/j2se/1.5.0//docs/api/javax/naming/directory/SearchControls.html).

Select the table in Model Explorer, then right-click and select New Child > Column from the context menu to create a new column.

Select the new column. The properties window will display the column's properties. Set the following properties:

Property	Value
Name	Enter a user-friendly name for the attribute.
Name in Source	Enter the LDAP attribute name that the column will represent. The attribute should be present in the base DN you've specified in the corresponding table's Name In Source field. The name must exactly match the attribute name in LDAP (though LDAP is not case sensitive).
Type	Enter String as datatype. String is required. Only string and byte array attribute values are supported by LDAP and BY JNDI. The LDAP Connector only supports string values. Support for byte arrays may be added in a future revision, if necessary.
Cardinality	Enter the number of "rows" (or nodes) contained within the modeled table. You can obtain this information later by issuing a query against a VDB that contains the model you are currently creating. An example query: select count(*) from people
Length	Enter the maximum known length of the string in the Length property. If there is no known maximum value, enter a reasonable default. Strings from LDAP that exceed this limit will not be truncated. If applications rely on column length metadata to perform operations, then be sure to use a length that is appropriate for those applications. You can obtain this information after the fact by submitting queries against each column to determine the maximum length. An example query: Select max(length(id)) from people
Distinct Value Count [Optional]	If you know the number of distinct values for this LDAP attribute, or have an estimation, then enter that number in the Distinct Value Count property. The Distinct Value Count information will be used in cost-based query analysis, to aid in increased query performance. You can obtain this value at a later time, by querying the VDB that contains this table and column to determine the distinct value count. An example query: select count(distinct(costcenter)) from people
Null Value Count [Optional]	If you know the number of null values for this LDAP attribute, or have an estimation, then enter that number in the Null Value Count property. The Null Value Count property will be used in cost-based query analysis. If there are no null values, enter –1. You can obtain this value at a later time, by querying the VDB that contains this table and column to determine the distinct value count. An example query: select count(*) from people where CostCenter=null

Property	Value
Maximum Value and Minimum Value [Optional]	If minimum and/or maximum values are known for this LDAP attribute, then enter that information in the Maximum Value and Minimum Value properties. This information will be used in cost-based query analysis. You can obtain this value at a later time, by querying the VDB that contains this table and column to determine the distinct value count. An example query: select max(convert(guid, integer)), min(convert(guid, integer)) from people Note: This assumes the column is always convertible from string to integer.

Create a new attribute and set the attribute's properties for each attribute you wish to model for your new table.

The entire process can be repeated for each base DN you wish to model:

- Create Table and set properties for the base DN
- Create the desired attributes in the Table and set properties for each attribute.

Check the Problems tab to fix any errors or warnings in the new model.

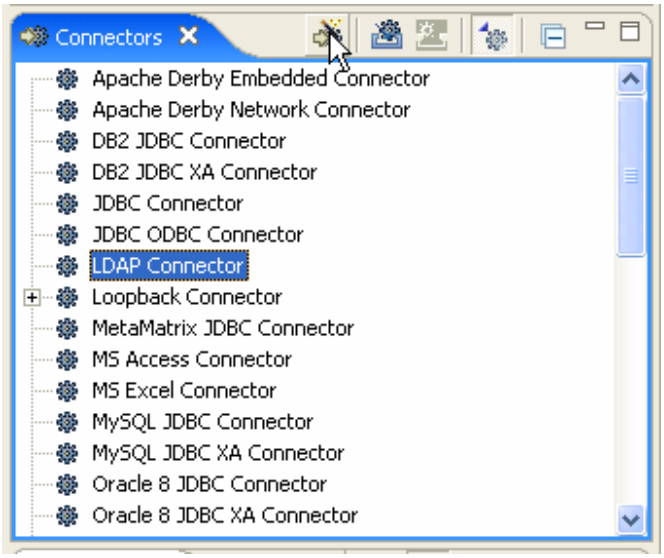
Save the model. You have successfully modeled an LDAP data source in a Relational model type.

NOTE: Periodically, the cost statistics information may need to be updated to reflect changes in the source system, in order to continue to receive good query plans. Be sure to take note of how often the data characteristics change in the LDAP system, and use this information to determine when these statistics may need to be updated.

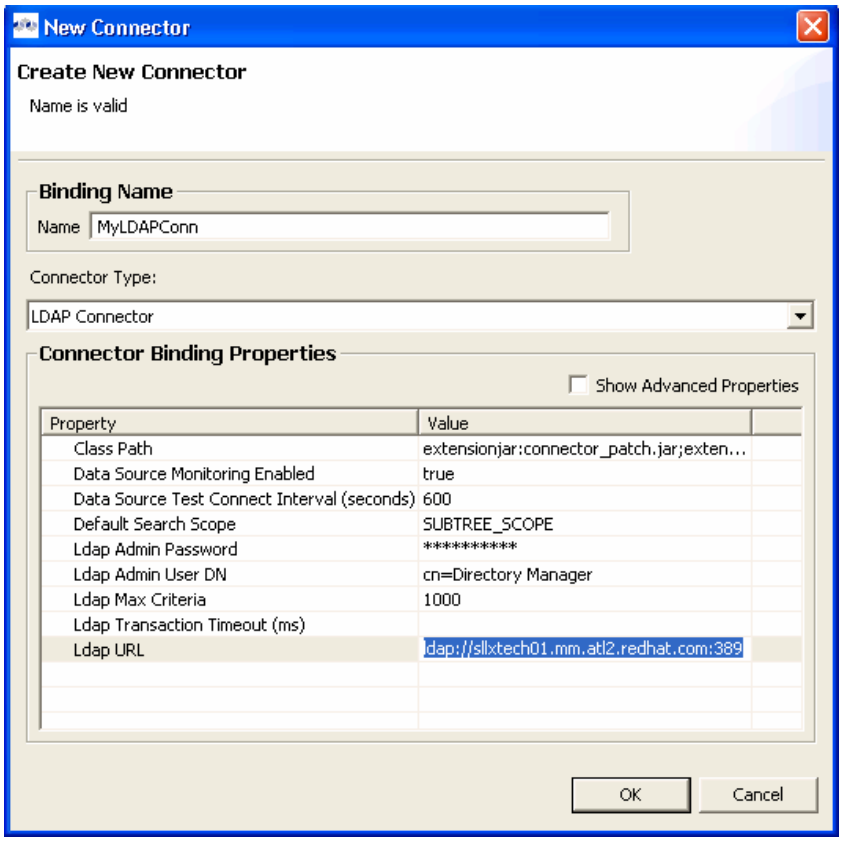
CREATING THE LDAP CONNECTOR INSTANCE

Creating and testing the LDAP Connector within Designer is very straightforward as outlined below. Please refer to the Enterprise Designer User's Guide for more details.

In the connectors view in Designer, select the LDAP Connector type, then click the "New Connector" button on the tool bar as shown.



This button will display the New Connector Wizard, below.



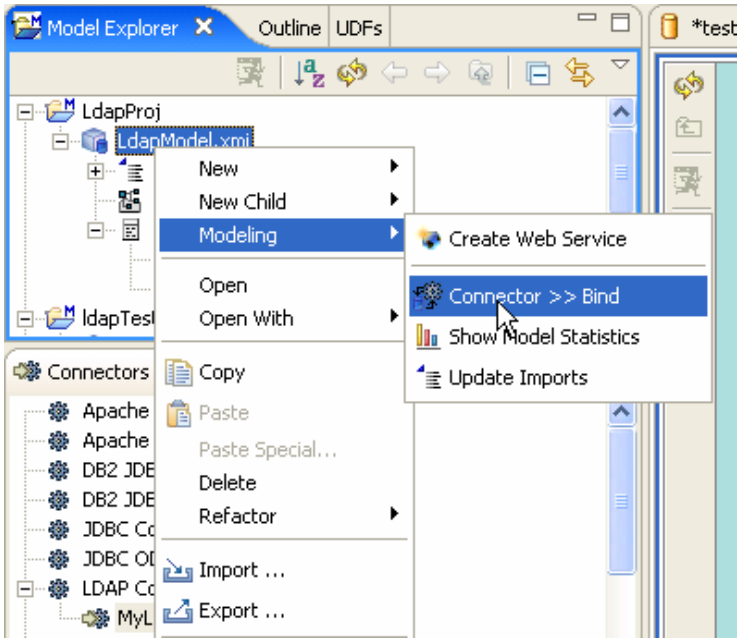
You can now set the properties for your new connector. The following table shows a summary of the available properties for the LDAP connector.

Property	Value
LDAP Admin User DN	User with which the connector will establish connection with LDAP.
LDAP Admin Password	Password associated with the Admin User DN.
LDAP Transaction Timeout (ms)	The default timeout (equivalent to the TCP timeout setting) can be overridden here. The units are in milliseconds.
LDAP Max Criteria	The maximum number of criteria in the where clause that will be passed to LDAP at any given time. If the LDAP server sets a limit on the number of criteria or size of a search clause, be sure to set the limit accordingly. 1000 is a reasonable default limit, though higher numbers should be used when the LDAP server supports it, and large where clauses may occur.
LDAP URL	The LDAP-specific URL of the LDAP server, e.g. "ldap://ldap-machine.company.com:389". Be sure to use "ldaps" if the connection must be performed over SSL. Also, if the connection is over SSL, be sure to import the server's certificate into the MetaMatrix certificate chain. For more information on SSL configurations, refer to the standard MetaMatrix documentation.
Maximum Result Rows	The maximum number of rows returned by the LDAP connector before an exception is thrown and no rows are returned.
Default Search Scope	Limited to OBJECT_SCOPE, ONELEVEL_SCOPE, or SUBTREE_SCOPE. Defaults to SUBTREE_SCOPE.
Restrict Searches to Named Object Class	Boolean value which, if set to 'true', uses the Base Table Name (not the Name In Source) as the restriction.
Default Search Base DN	The value to use as the default Search Base DN.
Connector Maximum Thread Count	Maximum number of threads created in the connector at one time. The setting for the connector max thread count should be at least twice as high as the "Connector Pool Max Connections" setting, in order to accommodate for 1 thread for each connection, and 1 thread for each execution. This way, the number of connections is not limited by the number of available threads. When configuring the binding in the MetaMatrix server, the setting for each MMProcess' "Max Threads" should be large enough to accommodate for the maximum number of connections in all connector pools. Check with the EII administrator for details; the MMProcess Max Threads is controlled in Configuration > Deployment section of the Console. If concurrency is high and throughput is too low, this number should be raised. If the LDAP server is overworked, this property may be lowered to reduce the load on the backend server.

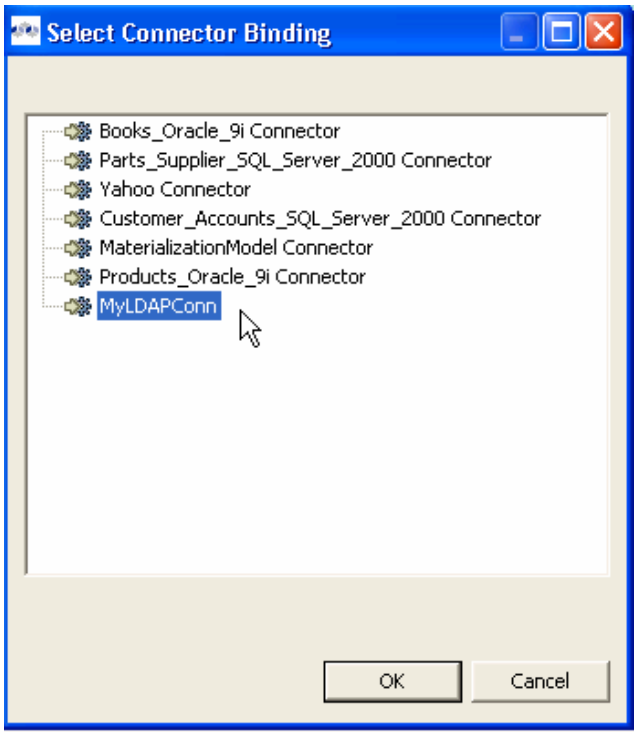
Property	Value
Connector Pool Cleaning Interval (Seconds)	Connector Pool Cleaning Interval is the interval between checking for idle connections. If a non-zero setting is specified, a thread will wake up at each interval and inspect each connection to see if it is not alive, or if it is alive, but unused for the Live and Unused Time or longer. If either condition is true, the connection will be removed from the pool. If the Connector Pool Cleaning Interval is significantly longer than the Live and Unused Time, then connections that have remained live and unused for longer than the Live and Unused Time may end up being re-used. The connection pool will still verify that the connection is alive before reusing it. However, if the Live and Unused Time is set according to an LDAP timeout, then it's likely that the connection will not be alive. Therefore, the Connector Pool Cleaning Interval should be relatively close to the Live and Unused Time, especially if the LDAP data source is known to timeout on old connections. If a setting of "0" is used, and the Wait For Source Time and Live and Unused Time are also set to zero, then connections in the pool will be persistent, and will remain in the pool until they fail.
Connector Pool Live and Unused Time (Seconds)	The Live and Unused Time is the time a connection is allowed to stay active and unused before it is considered closable. The actual amount of time a connection is allowed to stay alive and unused is less than or equal to Connector Pool Cleaning Interval + Live and Unused Time. This is a "worst-case" calculation that assumes the connection is returned to the pool at exactly the same time that the cleaning is performed, which means that the connection will be cleaned at the subsequent cleaning interval at worst. If the LDAP server has a timeout set to determine when to drop old connections, then the Live and Unused Time should be set to an amount that is less than or equal to that amount. In order for Live and Unused Time to go into effect, ENABLE_SHRINKING must be set to TRUE. The LDAP connection has a means of checking to see if it is still alive. In cases where the connection goes stale (e.g. the server was rebooted, or the network is unavailable), isAlive will fail, and a new connection will be attempted. If a setting of "0" is used, and the Wait For Source Time and Connector Pool Cleaning Interval are also set to zero, then connections in the pool will be persistent, and will remain in the pool until they fail.
Connector Pool Wait For Source Time (Seconds):	Pool Wait For Source Time is the maximum time a connector will wait for a connection to become available. In cases where the number of concurrent queries far outnumbers the Pool Max Connections, there may be a waiting period before a connection is obtained. If this is the case, then either Pool Max Connections should be increased, or Pool Wait For Source Time should be increased. In this situation, increasing Pool Max Connections will accommodate more concurrent queries. This will increase throughput, but will increase load on the backend LDAP server. If, instead, Pool Wait For Source Time is increased, more queries will be serviced before they timeout, and the load on the backend server will not increase, but the average query time will increase. If a setting of "0" is used, and the Live and Unused Time and Connector Pool Cleaning Interval are also set to zero, then connections in the pool will be persistent, and will remain in the pool until they fail.

Upon setting the LDAP connector properties and finishing the New Connector Wizard, the new LDAP connector will be displayed in the connectors view under the LDAP Connector type.

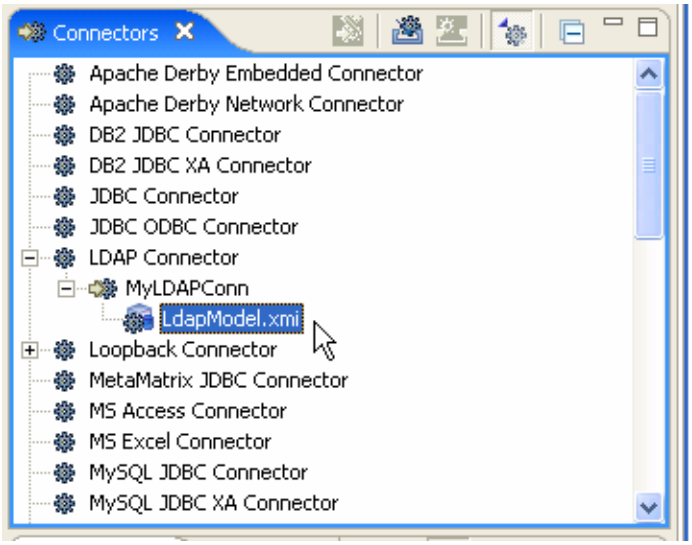
Next, you will need to specify that your LDAPModel will use the newly-created connector. Select the model in ModelExplorer, then rt-click and select Connector >> Bind as shown below.



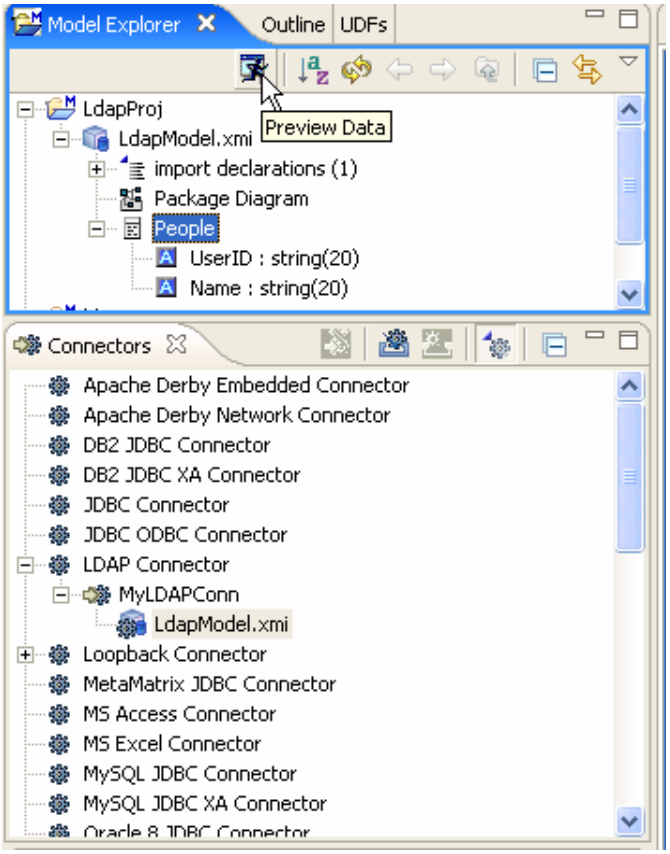
In the selection dialog, choose the LDAP Connector that you created previously:



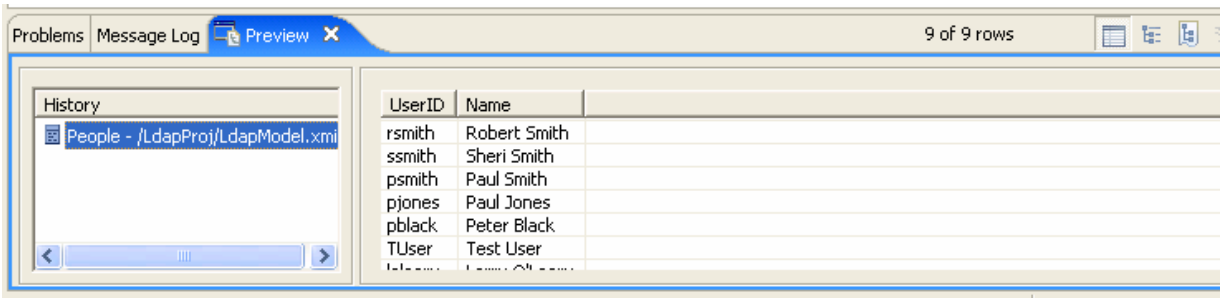
The model will now appear under the LDAP Connector in the connectors view, indicating that the model has been "bound" to the selected connector.



After the connector has been associated with the model, you are ready to preview your data. Select the table you want to preview in Model Explorer. The "Preview Data" icon should be enabled.



Selection of the "Preview Data" button will execute a query against the LDAP source. The results from the preview execution will show the first several rows of data for this table in the Preview tab:

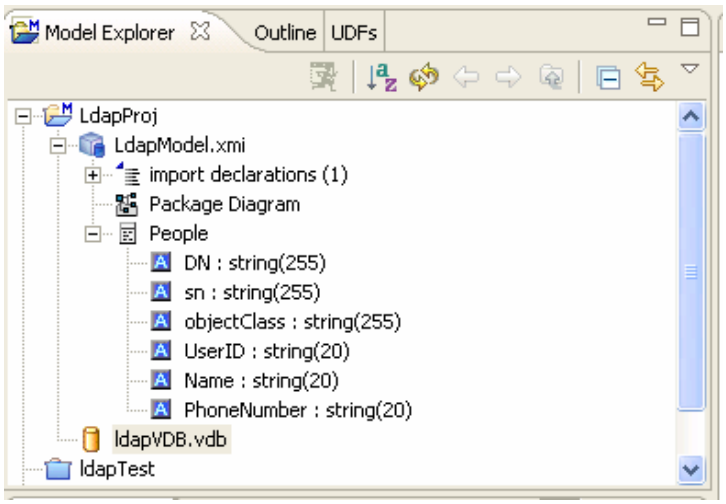


MODELING REQUIREMENTS FOR UPDATE CAPABILITY

The LDAP connector provides update capability, but additional modeling requirements are imposed beyond those required for read-only access. The additional requirements are described below, along with a few example screenshots.

- Table 'supports update' property – for each table in your LDAP source model that you want to enable updates, the 'supports update' property must be set to 'true'.
- Column 'Updateable' property – for each column in your LDAP source model that you want to enable updates, the 'Updateable' property must be set to 'true'
- Required Columns –
  - DN – The 'DN' (distinguished name) must be modeled for all update types (INSERT, UPDATE, and DELETE). For UPDATE and DELETE capability, the DN is the only required additional column. When executing an UPDATE or DELETE, the DN must be specified in the criteria. When executing an INSERT, the DN must be one of the columns specified in the INSERT.
  - objectClass – The 'objectClass' must always be specified for INSERT capability. When executing an INSERT, the objectClass must always be specified in the INSERT.
  - additional – each objectClass defined in the LDAP directory's schema may also have one or more additional required columns. Consult your LDAP server's documentation for information on determining what additional columns are required for each defined objectClass.

The following screenshot illustrates how the 'People' example used previously was modeled so that it could support updates. The column 'DN' (distinguished name) is required for all update types (INSERT, UPDATE, DELETE). The column 'objectClass' is required for INSERT. In this example, 'sn' (surname) is also required by the LDAP object, therefore is required. The 'PhoneNumber' column is an updateable column that is added for illustrating the update capability.



Once the LDAP source model changes shown above have been made, the model can be incorporated into a VDB and tested (see the Enterprise Designer User's Guide for more information on this process). The following are some example queries that illustrate the update capability:

SELECT

"SELECT * FROM LdapModel.People"

- Return all records in the LdapModel.People table.

INSERT

"INSERT INTO LdapModel.People (DN,sn,objectclass,Name) VALUES ('cn=Joe Young,ou=people,dc=metamatrix,dc=com','Young','person', 'Joe Young')"

- Insert a new object into the LDAP tree. 'DN' must be one of the columns specified in the INSERT. 'objectClass' must also be modeled and specified in the INSERT. 'sn' is also a requirement for this example's object.

UPDATE

"UPDATE LdapModel.People SET PhoneNumber='(314) 299-2999' WHERE DN='cn=Joe Young,ou=people,dc=metamatrix,dc=com'"

- Update the telephone number of the specified DN. 'DN' must be specified in the criteria.

DELETE

"DELETE FROM LdapModel.People WHERE DN='cn=Joe Young,ou=people,dc=metamatrix,dc=com'"

- Delete the specified DN from the LDAP tree. 'DN' must be specified in the criteria.

VDB CREATION AND TEST IN DESIGNER

You can now define a VDB in Designer using your LDAP model. Please refer to the Enterprise Designer User's Guide for more information about this process. The resulting VDB can be tested in Designer and can subsequently be deployed to a running MetaMatrix Server.

CONNECTOR DEPLOYMENT IN CONSOLE

The VDB created in Designer can be deployed to a running MetaMatrix Server. Please refer to the Enterprise Console User's Guide for more information about this process. Upon deployment or import of the VDB into the Console, the previously established LDAP connector will be imported with the VDB and will be available for use in the Console.

If desired, the user can export the connector information only from Designer, and import the connector by itself into the Console. Please refer to the Enterprise Designer and Console User's Guide for more information about the export and import processes.

Chapter 4: Connector Capabilities and Usage Considerations

CAPABILITIES SUPPORT

LDAP does not provide the same set of functionality as a relational database. The LDAP Connector supports many standard SQL constructs, and performs the job of translating those constructs into an equivalent LDAP search statement. For example, the SQL statement:

SELECT firstname, lastname, guid

FROM public_views.people

WHERE

(lastname='Jones' and firstname IN ('Michael', 'John'))

OR

guid > 600000

Uses a number of SQL constructs, including:

- SELECT clause support.
- select individual element support (firstname, lastname, guid).
- FROM support.
- WHERE clause criteria support.
- nested criteria support.
- AND, OR support
- Compare criteria (Greater-than) support.
- IN support.

The LDAP Connector executes LDAP searches by "pushing down" the equivalent LDAP search filter whenever possible, based on the supported capabilities.

MetaMatrix will automatically provide additional database functionality when the LDAP Connector does not explicitly provide support for a given SQL construct. In these cases, the SQL construct cannot be "pushed down" to the data source, so it will be evaluated in MetaMatrix, in order to ensure that the operation is performed.

In cases where certain SQL capabilities cannot be pushed down to LDAP, MetaMatrix will push down the capabilities that are supported, and fetch a set of data from LDAP. Then, MetaMatrix will evaluate the additional capabilities, creating a subset of the original data set. Finally, MetaMatrix will pass the result to the client.

It is useful to be aware of unsupported capabilities, in order to avoid fetching large data sets from LDAP when possible.

CAPABILITIES SUPPORT LIST

The following capabilities are supported in the LDAPConnector, and will be evaluated by LDAP

- SELECT queries.
- SELECT element pushdown (e.g. individual attribute selection)
- AND criteria.
- Compare criteria (e.g. <, <=, >, >=, =, !=).
- IN criteria.
- LIKE criteria.
- OR criteria
- INSERT, UPDATE, DELETE statements [must meet Modeling requirements]

Due to the nature of the LDAP source, the following capability is not supported:

- Stored Procedures.

The following capabilities are *not supported* in the LDAPConnector, and will be evaluated by the MetaMatrix system after data is fetched by the connector:

- Functions.
- Aggregates.
- BETWEEN Criteria.
- Case Expressions.
- Aliased Groups.
- Correlated Subqueries.
- EXISTS Criteria.
- Joins.
- Inline views.
- IS NULL criteria.
- NOT criteria.

- ORDER BY
- Quantified compare criteria.
- Row Offset.
- Searched Case Expressions.
- Select Distinct.
- Select Literals.
- UNION
- XA Transactions.

ATTRIBUTE DATATYPE SUPPORT

LDAP providers currently return attribute value types of java.lang.String and byte[], and do not support the ability to return any other attribute value type[3].

The LDAP Connector currently supports attribute value types of java.lang.String only. Therefore, all attributes should be modeled using the String datatype in MetaMatrix Designer.

If a model Designer wishes to convert a String value from LDAP into a different data type, they may do so using the many convenient conversion functions available in MetaMatrix – see the DevCentral website for details[4]. Some conversions may be applied *implicitly*, and do not require the use of any conversion functions. Other conversions must be applied *explicitly*, via the use of CONVERT functions.

Since the CONVERT functions are not supported by the underlying LDAP system, they will be evaluated in MetaMatrix. Therefore, if any criteria is evaluated against a converted datatype, that evaluation cannot be pushed to the data source, since the native type is String.

When converting from String to other types, be aware that criteria against that new data type will not be pushed down to the LDAP data source, which may decrease performance for certain queries. As an alternative, the data type can remain a string and the client application can make the conversion, or the client application can circumvent any

GREATER-THAN – LESS-THAN COMPARISONS

[3] http://java.sun.com/products/jndi/tutorial/ldap/faq/attr.html

[4] http://devcentral.metamatrix.com/tech/expl/DataTypeConversions

LDAP supports “>=” and “<=”, but has no equivalent for “>” or “<”. In order to support “<” or “>” pushdown to the source, the LDAP Connector will translate “<” to “<=”, and it will translate “>” to “>=”.

When using the LDAP Connector, be aware that strictly-less-than and strictly-greater-than comparisons will behave differently than expected. It is advisable to use “<=” and “>=” for queries against an LDAP-based data source, since this has a direct mapping to comparison operators in LDAP.

Chapter 5: Troubleshooting Common Problems

Testing Your Connector

The LDAP Connector relies on a number of properties that must accurately defined, or the MetaMatrix System will return unexpected results, or none at all.

This section describes the steps in deployment and data access wherein you might encounter difficulty deploying your MetaMatrix LDAP Connector.

As you deploy the connector in Console, improper configuration can lead to problems when you attempt to start your connector. You can test your LDAP Connector in MetaMatrix Enterprise Designer prior to Console deployment by submitting queries at modeling-time for verification.

Console Deployment Issues

The Console shows an Exception That Says Error Synchronizing the Server

If you receive an exception when you synchronize the server and your LDAP Connector is the only service that does not start, there was a problem starting the connector. Check to see that you have correctly typed in your connector properties.