Red Hat Directory Server 12

Configuring directory databases

Configuring and managing Red Hat Directory Server databases

Last Updated: 2024-05-07

Red Hat Directory Server 12 Configuring directory databases

Configuring and managing Red Hat Directory Server databases

Legal Notice

Abstract

You can configure databases, suffixes, chaining policy, database link, and referrals. Organize entries in custom groupings or hierarchies by using virtual directory trees.

Table of Contents

PROVIDING FEEDBACK ON RED HAT DOCUMENTATION

We appreciate your input on our documentation. Please let us know how we could make it better. To do so:

- For submitting feedback through Jira (account required):
  1. Log in to the Jira website.
  2. Click **Create** in the top navigation bar
  3. Enter a descriptive title in the **Summary** field.
  4. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
  5. Click **Create** at the bottom of the dialogue.
- For submitting feedback through Bugzilla (account required):
  1. Go to the Bugzilla website.
  2. As the Component, use **Documentation**.
  3. Fill in the **Description** field with your suggestion for improvement. Include a link to the relevant part(s) of documentation.
  4. Click **Submit Bug**.

CHAPTER 1. STORING SUFFIXES IN SEPARATE DATABASES

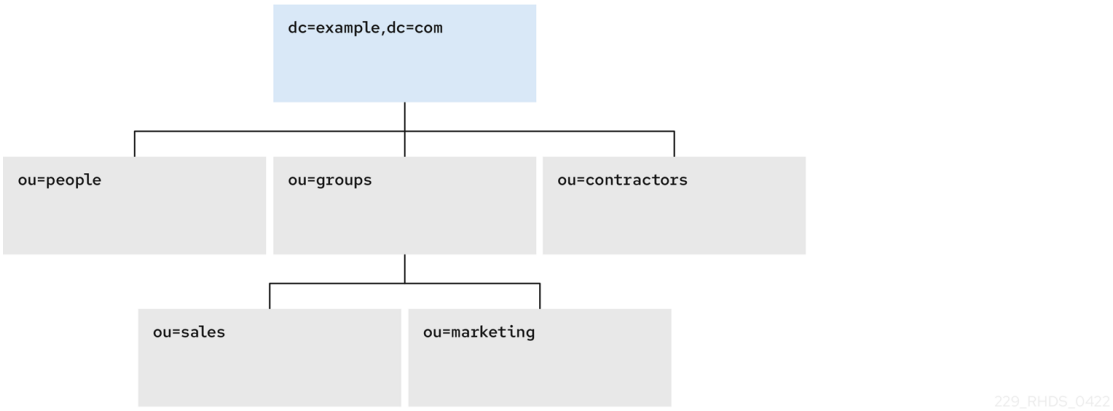
You can design distributed data storage logic in Directory Server by dividing data in an instance into several databases. You can use suffixes of directory trees as the method of data division.

You can create several directory trees and store them in separate databases by root suffixes. You can also divide a single directory tree into branches and store the branches in separate databases by sub-suffixes.

1.1. ROLE OF A SUFFIX IN THE DATA STRUCTURE

Directory Server presents data in hierarchical structures called directory trees (DIT). The following is a simple directory tree:

Figure 1.1. Simple directory tree with a single root suffix



Each directory tree has a single root entry which defines the naming context of that directory tree, such as **dc=example,dc=com**.

You can store various pieces of a directory tree in different databases, and then distribute these databases across multiple servers.

You can use suffixes to define the distribution logic of your data storage. A suffix associates a branch (subtree) of the directory tree with a particular database.

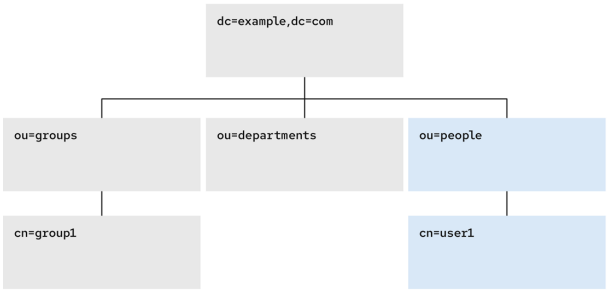
This way you can have multiple databases in a single instance of the server. You are not confined to a single database.

1.2. ROOT SUFFIX VS. SUB-SUFFIXES

A root suffix associates an entire directory tree (DIT) with a database. The root suffix does not have a parent suffix.

When you want to store a branch of a directory tree in a separate database, you create a sub-suffix, which associates the branch of the tree with a different database than ancestors of the branch. A sub-suffix must be attached to a parent suffix. The parent suffix can be the root suffix or a sub-suffix, which means that a branch of any subtree can be stored in a separate database.

Figure 1.2. Directory tree with a sub-suffix in a separate database



In this example, the **ou=people,dc=example,dc=com** sub-suffix is stored in one database and the rest of the directory tree under the root suffix is stored in a different database.

Advantages of using sub-suffixes:

- You can perform database maintenance (import/export/indexing) at a granular level.
- You can store sub-suffixes on separate disks, which resolves disk space concerns.

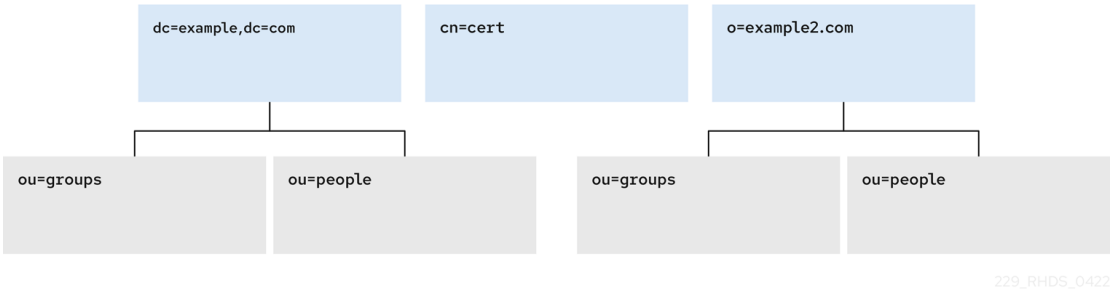
Disadvantages of using sub-suffixes:

- You need more management actions during setup time.
- Replication needs a separate configuration and replication agreement for each sub-suffix.

1.3. SEVERAL ROOT SUFFIXES

You can also have several directory trees (DIT) with different root suffixes in a single instance. For example, when you want to separate some portions of data from the user root.

Figure 1.3. Several directory trees defined by root suffixes



When clients search the **dc=example,dc=com** tree, the search does not return entries from the other trees, because they are off limits to the searching algorithm.

You can then choose which directory tree and naming context is default for your instance.

1.4. CREATING A ROOT SUFFIX USING THE COMMAND LINE

This procedure instructs you how to create the root suffix of a directory tree on the command line.

Procedure

1. Optional: List the suffixes and back-end databases that are already in use:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userroot)
```

The name in parentheses is the back-end database that stores the data of the corresponding suffix. You cannot use an existing database name when you create the root suffix in the next step.

2. Specify the DN of the root suffix in the **--suffix** argument and associate it with a new database using the **--be-name** argument:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend create --suffix="dc=example,dc=net" --be-name="example"
```

Verification

- List the suffixes and databases using the command from the first step of this procedure.

1.5. CREATING A ROOT SUFFIX USING THE WEB CONSOLE

This procedure instructs you how to create the root suffix of a directory tree in a browser.

Prerequisites

- You are logged in to the instance in the web console.

Procedure

1. Under **Database**, click the **Create Suffix** button below the configuration tree.
2. Fill in the **Suffix DN** and **Database Name**.
3. Select **Create The Top Suffix Entry** and click **Create Suffix**.

Verification

- The new suffix should appear in the tree of suffixes.

1.6. CHANGING THE DEFAULT NAMING CONTEXT

A naming context is an attribute of a directory tree (DIT) that defines the root namespace for entries in that DIT. When you structure data in your instance with multiple root suffixes, your instance has several DITs, each with a different naming context.

This procedure instructs you how to change the default naming context on the command line when you work with multiple root suffixes in your instance.

Clients that access your instance, may not know which naming context they need to use. The Directory Server signals to clients what the default naming context is, if they have no other configuration of a naming context known to them.

You set the default naming context in the **nsslapd-defaultnamingcontext** attribute in **cn=config**. Directory Server propagates this value over to the Directory Server Agent Service Entry (root DSE) and clients can query it anonymously.

Prerequisites

- You have created the root suffix that defines the default naming context of your instance.

Procedure

1. Optional: View the current default naming context:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com config get nsslapd-defaultnamingcontext
nsslapd-defaultnamingcontext: dc=example,dc=com
```

2. Replace the value of the **nsslapd-defaultnamingcontext** parameter with the required naming context:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com config replace nsslapd-defaultnamingcontext=dc=example,dc=net
```

Verification

- View the current default naming context. The value should be updated.

Additional resources

- Creating a root suffix using the command line
- Creating a root suffix using the web console

1.7. CREATING A SUB-SUFFIX USING THE COMMAND LINE

You can create a sub-suffix of a directory tree using the command line.

Prerequisites

- You created the parent suffix for the sub-suffix.

Procedure

1. Optional: List the suffixes and back-end databases that are already in use:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userroot)
```

The name in parentheses is the back-end database that stores the data of the corresponding suffix. You cannot use an existing database name when you create the sub-suffix in the next step.

2. Specify the full DN of the sub-suffix in the **--suffix** argument, associate it with a new database using the **--be-name** argument, and specify the parent suffix in the **--parent-suffix** argument:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend create --suffix="ou=People,dc=example,dc=com" --be-name="example" --parent-suffix="dc=example,dc=com" --create-suffix
```

With **--create-suffix** argument, the command creates the configuration entry for the sub-suffix and the sub-suffix entry **ou=People,dc=example,dc=com**.

The **--create-suffix** argument supports the creation of suffixes with the following RDN attribute types: **c**, **cn**, **dc**, **o**, and **ou**. If you want to create a suffix with RDN such as **l**, you can use the **dsconf backend create** command without the **--create-suffix** option and then add the suffix entry by using an LDAP add operation or importing the entry from an LDIF file.

Verification

- List the suffixes and databases using the command from the first step of this procedure.

1.8. CREATING A SUB-SUFFIX USING THE WEB CONSOLE

This procedure instructs you how to create a sub-suffix of a directory tree in a browser.

Prerequisites

- You are logged in to the instance in the web console.
- You created the parent suffix for the sub-suffix.

Procedure

1. Under **Database**, select a suffix from the configuration tree that is the parent of the sub-suffix.
2. Click the **Suffix Tasks** and select **Create Sub-Suffix**.
3. Fill in the **Sub-Suffix DN**, such as ***ou=People***, and **Database Name**.
4. Select **Create The Top Suffix Entry** and click **Create Sub-Suffix**.

Verification

- The new sub-suffix should appear among suffixes in the configuration tree.

CHAPTER 2. USING VIEWS TO CREATE A VIRTUAL DIRECTORY HIERARCHY

You can create virtual directory-tree (DIT) views to organize entries in custom groupings or hierarchies and thus navigate the standard DIT from various perspectives. This way you can save costs on management of your directory, and make navigation through entries more intuitive to the users of your service.

2.1. ABOUT VIEWS

Virtual directory-tree views, or *views*, are an optional layer of structure in addition to your standard directory tree (DIT) to categorize and search entries in your DIT.

Using views, you can create virtual directory hierarchies, so it is easy to navigate entries, regardless of their placement in the standard DIT. A view uses attributes of entries to include them in the virtual hierarchy, similarly to members of a filtered role or a dynamic group. To client applications, views appear as ordinary container hierarchies.

This way, you can initially place entries in a flat DIT and use views to categorize the entries in complex hierarchies without the need to move the entries. Additionally, entries can appear in multiple views, which you cannot achieve with a standard DIT.

You can think of views as *named filters*. Each view is an entry of the **nsView** object class and may have the **nsViewFilter** attribute, which says what entries are visible in that view. It may be desirable to restrict the type of entries to be returned by specifying an object class in the filter.

You can use a view as a container of other views and thus create the virtual hierarchy. A nested view inherits filters from its ancestors and restricts the view by combining its filter and ancestor filters with an **AND**, such as **(&(container filter)(view filter))**.

When a search is performed with a view as the base, entries that match the filter are returned from this virtual search space. The entries only appear to be nested under the view virtually, but they can actually be stored at any position in the DIT.

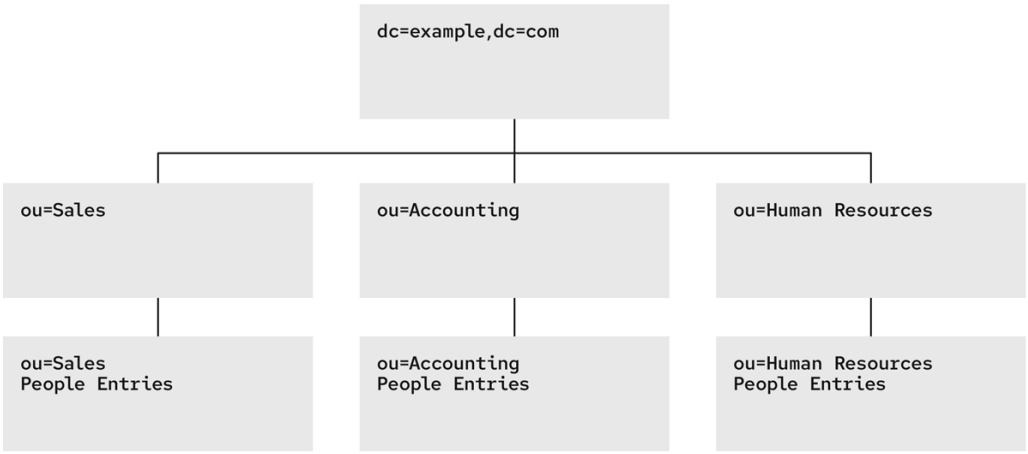
NOTE

You can create a testing instance and explore how views work on data imported from the example file located in **/usr/share/dirsrv/data/Example-views.ldif**.

2.2. DIRECTORY DESIGN CONSIDERATIONS

When you design a directory tree (DIT), you naturally tend to categorize entries with hierarchies to reflect hierarchies in your organization. A standard DIT without views ties the position of an entry in the DIT to the distinguished name (DN) of the entry and therefore it is more suitable for use with fixed hierarchies.

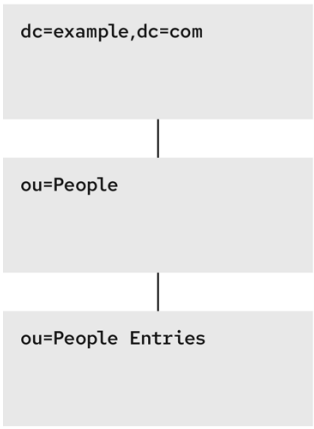
Figure 2.1. Standard hierarchy DIT based on functional units



However, the nature of hierarchies in an organization is dynamic. Moving an entry in a standard DIT is time-consuming, because with every change of the position of the entry, the entry and all its descendants must also be renamed. This leads to service disruptions and additional expenses, especially in changes of top-level subtrees.

It is a good idea to design a flat hierarchy with categorization of resources by characteristics that do not change, such as the resource type (people, equipment, etc.), and capture this hierarchy in a standard flat DIT.

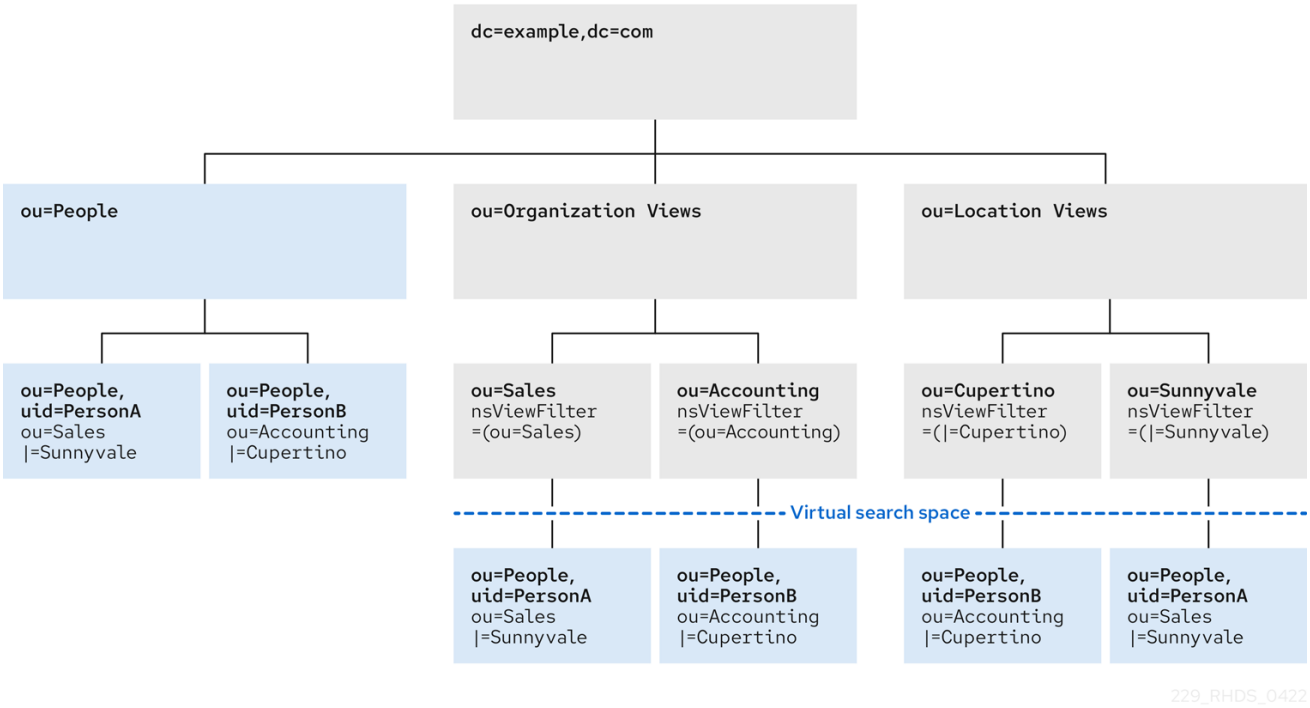
Figure 2.2. Standard flat DIT based on resource types



However, a flat DIT is not convenient for navigating the resources. Different users need to navigate the resources from different perspectives, such as an association with a functional unit or a geographical location, which would require additional tools or complex search queries in case of a flat DIT.

The solution to overcome the limitations of a flat DIT is provided by virtual hierarchies of views. Views allow creation of flexible hierarchies by separating the name of an entry from the position of the entry in the hierarchy. The virtual hierarchies are based on attributes instead.

Figure 2.3. DIT With Virtual Hierarchies of Views



2.3. BENEFITS OF USING VIEWS

Using virtual directory-tree views has the benefits of custom flexible hierarchies that are more intuitive for users to navigate and for administrators more efficient to maintain than a deeply-nested standard directory tree (DIT).

Flat and flexible naming

Views facilitate the use of a flat namespace for entries, because virtual DIT views provide navigational and managerial support similar to those provided by traditional hierarchies.
Whenever there is a change to the DIT, the entries never need to be moved; only the virtual DIT view hierarchies change. Because these hierarchies do not contain actual entries, they are simple and quick to modify.

Reduction of costs in case of design errors

Oversights during deployment planning are less catastrophic with virtual DIT views. If the hierarchy is not developed correctly in the first instance, it can be changed easily and quickly without disrupting the service.

Fast and cheap maintenance

View hierarchies can be completely revised in minutes and the results instantly realized, significantly reducing the cost of directory maintenance.
Changes to a virtual DIT hierarchy are instantly realized. When an organizational change occurs, a new virtual DIT view can be created quickly. The new virtual DIT view can exist at the same time as the old view, thereby facilitating a more gradual changeover for the entries themselves and for the applications that use them. Because an organizational change in the directory is not an all-or-nothing operation, it can be performed over a period of time and without service disruption.

Enhanced overall flexibility

Using multiple virtual DIT views for navigation and management allows for more flexible use of the directory service.

With the functionality provided by virtual DIT views, an organization can use both the old and new methods to organize directory data without any requirement to place entries at certain positions in the DIT.

Intuitive user navigation

Views promote flexibility in working practices and reduce the requirement that directory users create complex search filters, using attribute names and values that they would otherwise have no need to know.

The flexibility of having more than one way to view and query directory information allows end users and applications to find what they need intuitively through hierarchical navigation.

Shortcut to frequent search queries

Virtual DIT view hierarchies can be created as a kind of ready-made queries to facilitate the retrieval of frequently-required information.

2.4. COMPATIBILITY OF VIEWS WITH OTHER FEATURES

When working with views, the search space is limited to entries under a single suffix. Users must base their search queries on a view to get results from a virtual hierarchy. You must take a slightly different approach to access control. You can use entry grouping with roles and classes of service in views.

Multiple back ends

Virtual DIT views are not entirely compatible with multiple back ends.

The search is limited to a single back end, which means that the entries to be returned by the views must reside under the same suffix.

Search space

The virtual search space is separate from the standard search space. The virtual search space is accessible only when a search is based on a view node with a filter. Otherwise it is a conventional search over the standard directory tree (DIT) that does not return entries contained under virtual DIT hierarchies.

For example, a search based on **dc=example,dc=com** does not return any entries from the virtual search space of views; in fact, no virtual-search-space search is performed. Views processing occurs if the search base is such as **ou=Cupertino,ou=Location Views,dc=example,dc=com**.

This way, Directory Server ensures that the search does not result in entries from both places.

Access control

The use of views requires a slightly different approach to access control. Because there is currently no explicit support for access control lists (ACL) in views, create role-based ACL at the view parent and add the roles to the appropriate parts of the view hierarchy. In this way, take advantage of the organizational property of the hierarchy.

Entries grouping

Both *class of service* and *roles* in Directory Server support views. When adding a *class of service* or a *role* under a view hierarchy, the entries that are both logically and actually contained in the view are considered within scope. This means that *roles* and *class of service* can be applied using a virtual DIT view, but the effects of that application can be seen even when querying the flat namespace.

2.5. COMPATIBILITY OF VIEWS WITH CLIENT APPLICATIONS

Virtual directory tree (DIT) views are designed to mimic standard DITs to a high degree. The existence

of views should be transparent to most applications; there should be no indication that they are working with views. Except for a few specialized cases, there is no need for directory users to know that views are being used in a Directory Server instance; views appear and behave like standard DITs.

Certain types of applications may have problems working with a views-enabled directory service. For example:

- Applications that use the distinguished name (DN) of a target entry to navigate up the DIT.
  This type of application would find that it is navigating up the hierarchy in which the entry physically exists instead of the view hierarchy in which the entry was found. The reason for this is that views make no attempt to disguise the true location of an entry by changing the DN of the entry to conform to the view's hierarchy.

  This is by design - many applications would not function if the true location of an entry were disguised, such as those applications that rely on the DN to identify a unique entry. This upward navigation by deconstructing a DN is an unusual technique for a client application, but, nonetheless, those clients that do this may not function as intended.

- Applications that use the **numSubordinates** operational attribute to determine how many entries exist beneath a node.
  For the nodes in a view, this is currently a count of only those entries that exist in the standard search space, ignoring the virtual search space. Consequently, applications may not evaluate the view in a search.

2.6. CREATING A VIEW

This procedure instructs you how to create a virtual directory-tree view on the command line.

Procedure

- Add a view entry with the **ldapadd** utility.
  Specify the **nsView** object class and define a view filter in the **nsViewFilter** attribute:

```
# ldapadd -D "cn=Directory Manager" -W -H ldap://server.example.com -x
dn: ou=PeopleInRoom0466,dc=example,dc=com
objectClass: top
objectClass: organizationalUnit
objectClass: nsView
ou: PeopleInRoom0466
description: People in the room 0466
nsViewFilter: (&(objectClass=inetOrgPerson)(roomNumber=0466))
```

Verification

- Perform a search with the view as the search base:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -x -b
ou=PeopleInRoom0466,dc=example,dc=com
```

2.7. CREATING INDEXES TO IMPROVE THE PERFORMANCE OF VIEWS USING THE COMMAND LINE

Views are derived from search results based on a given filter. Part of the filter are the attributes given explicitly in the **nsViewFilter**; the rest of the filter is based on the entry hierarchy, looking for the **entryid** and **parentid** operational attributes of the actual entries included in the view.

```
(|(parentid=search_base_id)(entryid=search_base_id)
```

If any of the searched attributes – **entryid**, **parentid**, or the attributes in the **nsViewFilter** – are not indexed, then the search becomes partially unindexed and Directory Server searches the entire directory tree for matching entries.

To improve views performance, create the indexes as follows:

- Create *equality index* (**eq**) for **entryid**. The **parentid** attribute is indexed in the system index by default.
- If a filter in **nsViewFilter** tests presence (**attribute=***), then create *presence index* (**pres**) for the attribute being tested. You should use this index type only with attributes that appear in a minority of directory entries.
- If a filter in **nsViewFilter** tests equality (**attribute=value**), create *equality index* (**eq**) for the attribute being tested.
- If a filter in **nsViewFilter** tests a substring (**attribute=value***), create *substring index* (**sub**) for the attribute being tested.
- If a filter in **nsViewFilter** tests approximation (**attribute~=value**), create *approximate index* (**approximate**) for the attribute being tested.

For example, when you use the following view filter:

```
nsViewFilter: (&(objectClass=inetOrgPerson)(roomNumber=*66))
```

you should index **objectClass** with the *equality index*, which is done by default, and **roomNumber** with the *substring index*.

Prerequisites

- You are aware of the attributes that you use in a view filter.

Procedure

1. Optional: List the back ends to determine the database to index:

   ```
   # dsconf -D "cn=Directory Manager" instance_name backend suffix list
   dc=example,dc=com (userroot)
   ```

   Note the selected database name (in parentheses).

2. Create index configuration with the **dsconfig** utility for the selected back-end database. Specify the attribute name, index type, and, optionally, matching rules to set collation order (OID), especially in case of an internationalized instance.

   ```
   # dsconf -D "cn=Directory Manager" instance_name backend index add --attr
   roomNumber --index-type sub userroot
   ```

Repeat this step for each attribute used in the view filter.

3. Reindex the database to apply the new indexes:

```
# dsconf -D "cn=Directory Manager" instance_name backend index reindex userroot
```

Verification

1. Perform a search that is based on the standard directory tree with the same filter that you use in the view:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -x -b
dc=example,dc=com (&(objectClass=inetOrgPerson)(roomNumber=*66))
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -x -b
dc=example,dc=com "(&(objectClass=inetOrgPerson)(roomNumber=*66))"
```

2. View the *access* log in **/var/log/dirsrv/slapd-*instance_name*/access**.
The **RESULT** of your search should not contain **note=U** or **Partially Unindexed Filter** in the details.

Additional resources

- Managing indexes

2.8. CREATING INDEXES TO IMPROVE THE PERFORMANCE OF VIEWS USING THE WEB CONSOLE

Views are derived from search results based on a given filter. Part of the filter are the attributes given explicitly in the **nsViewFilter**; the rest of the filter is based on the entry hierarchy, looking for the **entryid** and **parentid** operational attributes of the actual entries included in the view.

```
(|(parentid=search_base_id)(entryid=search_base_id)
```

If any of the searched attributes – **entryid**, **parentid**, or the attributes in the **nsViewFilter** – are not indexed, then the search becomes partially unindexed and Directory Server searches the entire directory tree for matching entries.

To improve views performance, create the indexes as follows:

- Create *equality index* (**eq**) for **entryid**. The **parentid** attribute is indexed in the system index by default.
- If a filter in **nsViewFilter** tests presence (**attribute=***), then create *presence index* (**pres**) for the attribute being tested. You should use this index type only with attributes that appear in a minority of directory entries.
- If a filter in **nsViewFilter** tests equality (**attribute=value**), create *equality index* (**eq**) for the attribute being tested.
- If a filter in **nsViewFilter** tests a substring (**attribute=value***), create *substring index* (**sub**) for the attribute being tested.
- If a filter in **nsViewFilter** tests approximation (**attribute~=value**), create *approximate index* (**approximate**) for the attribute being tested.

For example, when you use the following view filter:

```
nsViewFilter: (&(objectClass=inetOrgPerson)(roomNumber=*66))
```

you should index **objectClass** with the *equality index*, which is done by default, and **roomNumber** with the *substring index*.

Prerequisites

- You are logged in to the instance in the web console.
- You are aware of the attributes that you use in a view filter.

Procedure

1. Under **Database**, select a suffix from the configuration tree for which you want to create an index.
2. Navigate to **Indexes** and **Database Indexes**.
3. Click the **Add Index** button.
4. Type the name of the attribute and select the attribute.
5. Select the **Index Types** that should be created for this attribute.
6. Optionally, add **Matching Rules** to specify collation order (OID), especially in case of an internationalized instance.
7. Select **Index attribute after creation** to rebuild the index afterwards.
8. Click **Create Index**.
9. Repeat the steps for each attribute to be indexed.

Verification

- **Filter Indexes** by typing the name of the added attribute.
- The newly indexed attribute should appear in the results.

Additional resources

- Managing indexes

CHAPTER 3. SWITCHING A DATABASE TO READ-ONLY MODE

Databases of Directory Server run in read-write mode by default, in which users can both retrieve and store data.

When you need a faithful image of a database at a given time, for example before a backup or before a manual initialization of a consumer, you may switch a database to read-only mode that prevents users from creating, modifying, or deleting entries.

3.1. PREREQUISITES

- The database is in read-write mode.
- The database is not used in replication, since enabling read-only mode disables replication.

3.2. SWITCHING A DATABASE TO READ-ONLY MODE USING THE COMMAND LINE

This procedure instructs you how to switch a Directory Server database to read-only mode on the command line.

Procedure

1. List the suffixes and their corresponding databases:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userroot)
o=test (test_database)
```

   Note the name or suffix of the database that you want to switch.

2. Enable read-only mode with the **--enable-readonly** parameter and specify the database either by name or suffix:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix set --enable-readonly "test_database"
```

Verification

- Attempt a write operation to the directory, such as:

```
# ldapmodify -D "cn=Directory Manager" -W -H ldap://server.example.com -x
dn: dc=example,dc=com
changetype: modify
add: description
description: foo
```

  The server should refuse to perform.

```
modifying entry "dc=example,dc=com"
ldap_modify: Server is unwilling to perform (53)
 additional info: Server is read-only
```

Additional resources

- Switching an entire instance to read-only mode

3.3. SWITCHING A DATABASE TO READ-ONLY MODE USING THE WEB CONSOLE

This procedure instructs you how to switch a Directory Server database to read-only mode in a browser.

Prerequisites

- You are logged in to the instance in the web console.

Procedure

1. Under **Database**, select the suffix in the configuration tree.
2. Check the **Database Read-Only Mode** option.
3. Click **Save Configuration**.

Verification

- Attempt a write operation to the directory, such as:

```
# ldapmodify -D "cn=Directory Manager" -W -H ldap://server.example.com -x
dn: dc=example,dc=com
changetype: modify
add: description
description: foo
```

The server should refuse to perform.

```
modifying entry "dc=example,dc=com"
ldap_modify: Server is unwilling to perform (53)
 additional info: Server is read-only
```

3.4. ADDITIONAL RESOURCES

- Backup up Directory Server

CHAPTER 4. SWITCHING AN INSTANCE TO READ-ONLY MODE

By default, instances run in read-write mode, in which users can both retrieve and store data. In emergency cases, such as when you want to prevent replication or disable modification of data during reindexing, but keep the directory available, you can temporarily switch the instance to read-only mode.

If Directory Server maintains more than one database and all databases need to be switched to read-only, you can do this in a single operation, on the command line or in the web console.

WARNING

In read-only mode, you cannot restart the instance, but you may still modify the configuration.

If you stop an instance in read-only mode, you cannot start it again until you manually disable read-only mode.

To disable read-only mode manually, open the **/etc/dirsrv/slapd-*instance_name*/dse.ldif** file, navigate to the **cn=config** section, and set the **nsslapd-readonly** parameter to **off**.

4.1. PREREQUISITES

- The instance is in read-write mode.
- The instance is not used in replication, since enabling read-only mode disables replication.

4.2. SWITCHING AN INSTANCE TO READ-ONLY MODE USING THE COMMAND LINE

This procedure instructs you how to switch a Directory Server instance to read-only mode on the command line.

Procedure

- Set the **nsslapd-readonly** parameter to **on**:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com config replace
nsslapd-readonly=on
```

Verification

- Attempt a write operation to the directory, such as:

```
# ldapmodify -D "cn=Directory Manager" -W -H ldap://server.example.com -x
dn: dc=example,dc=com
changetype: modify
```

```
add: description
description: foo
```

The server should refuse to perform.

```
modifying entry "dc=example,dc=com"
ldap_modify: Server is unwilling to perform (53)
 additional info: Server is read-only
```

Additional resources

- Switching a database to read-only mode

4.3. SWITCHING AN INSTANCE TO READ-ONLY MODE USING THE WEB CONSOLE

This procedure instructs you how to switch a Directory Server instance to read-only mode in a browser.

Prerequisites

- You are logged in to the instance in the web console.

Procedure

1. Under **Server**, select the **Advanced Settings** tab.
2. Check the **Server Read-Only** option.
3. Click **Save**.

Verification

- Attempt a write operation to the directory, such as:

```
# ldapmodify -D "cn=Directory Manager" -W -H ldap://server.example.com -x
dn: dc=example,dc=com
changetype: modify
add: description
description: foo
```

The server should refuse to perform.

```
modifying entry "dc=example,dc=com"
ldap_modify: Server is unwilling to perform (53)
 additional info: Server is read-only
```

Additional resources

- Switching a database to read-only mode

CHAPTER 5. DELETING A DATABASE OF A SUFFIX THAT IS NO LONGER NEEDED

When you need to reclaim disk space on your Directory Server host, you can delete databases of suffixes that are not in use anymore.

5.1. DELETING A DATABASE USING THE COMMAND LINE

This procedure instructs you how to delete a Directory Server database on the command line.

Procedure

1. List suffixes and their corresponding databases:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userroot)
o=test (test_database)
```

   Note the name of the database that you want to delete.

2. Enter the **dsconf backend delete** command and specify the name of the database:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend delete
"test_database"
```

3. Confirm the deletion by typing "Yes I am sure" in the prompt:

```
Deleting Backend cn=test_database,cn=ldbm database,cn=plugins,cn=config :
Type 'Yes I am sure' to continue: Yes I am sure
```

Verification

- List the suffixes/databases:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userroot)
```

5.2. DELETING A DATABASE USING THE WEB CONSOLE

This procedure instructs you how to delete a Directory Server database in a browser.

Prerequisites

- You are logged in to the instance in the web console.

Procedure

1. Under **Database**, select the suffix that you want to delete.
2. Navigate to **Suffix Tasks → Delete Suffix**.
3. Select **Yes, I am sure.**.

4. Click **Delete Suffix**.

Verification

- Under **Database**, review the list of suffixes in the configuration tree.

CHAPTER 6. VERIFYING THE INTEGRITY OF BACK-END DATABASES

The Directory Server database integrity check can detect problems, such as corrupt metadata pages and the sorting of duplicate keys. If problems are found, you can, depending on the problems, re-index the database or restore a backup.

6.1. PERFORMING A DATABASE INTEGRITY CHECK

The **dsctl dbverify** command enables administrators to verify the integrity of back-end databases.

Procedure

1. Optional: List the back-end databases of the instance:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend suffix list
dc=example,dc=com (userRoot)
```

2. Stop the instance:

```
# dsctl instance_name stop
```

3. Verify the database. For example, to verify the **userRoot** database, enter:

```
# dsctl instance_name dbverify userRoot
[04/Feb/2022:13:11:02.453624171 +0100] - INFO -
ldbm_instance_config_cachememsize_set - force a minimal value 512000
[04/Feb/2022:13:11:02.465339507 +0100] - WARN -
ldbm_instance_add_instance_entry_callback - ldbm instance userroot already exists
[04/Feb/2022:13:11:02.468060144 +0100] - ERR - ldbm_config_read_instance_entries -
Failed to add instance entry cn=userroot,cn=ldbm database,cn=plugins,cn=config
[04/Feb/2022:13:11:02.471079045 +0100] - ERR - bdb_config_load_dse_info - failed to read
instance entries
[04/Feb/2022:13:11:02.476173304 +0100] - ERR - libdb - BDB0522 Page 0: metadata page
corrupted
[04/Feb/2022:13:11:02.481684604 +0100] - ERR - libdb - BDB0523 Page 0: could not check
metadata page
[04/Feb/2022:13:11:02.484113053 +0100] - ERR - libdb - /var/lib/dirsrv/slapd-
instance_name/db/userroot/entryrdn.db: BDB0090 DB_VERIFY_BAD: Database verification
failed
[04/Feb/2022:13:11:02.486449603 +0100] - ERR - dbverify_ext - verify failed(-30970):
/var/lib/dirsrv/slapd-instance_name/db/userroot/entryrdn.db
dbverify failed
```

4. If the verification process reported any problems, fix them manually or restore a backup.

5. Start the instance:

```
# dsctl instance_name start
```

Additional resources

- Restoring Directory Server

CHAPTER 7. MANAGING ATTRIBUTE ENCRYPTION

Directory Server offers a number of mechanisms to secure access to sensitive data in the directory. However, by default, the server stores data unencrypted in the database. For highly sensitive information, the potential risk that an attacker could gain access to the database, can be a significant risk.

The attribute encryption feature enables administrators to store specific attributes with sensitive data, such as government identification numbers, encrypted in the database. When enabled for a suffix, every instance of these attributes, even the index data, is encrypted for every entry stored in this attribute in the database. Note that you can enable attribute encryption for suffixes. To enable this feature for the whole server, you must enable attribute encryption for each suffix on the server. Attribute encryption is fully compatible with **eq** and **pres** indexing.

IMPORTANT

Any attribute you use within the entry distinguished name (DN) cannot be efficiently encrypted. For example, if you have configured to encrypt the **uid** attribute, the value is encrypted in the entry, but not in the DN:

```
dn: uid=demo_user,ou=People,dc=example,dc=com
...
uid::Sf04P9nJWGU1qiW9JJCGRg==
```

7.1. KEYS DIRECTORY SERVER USES FOR ATTRIBUTE ENCRYPTION

To use attribute encryption, you must configure encrypted connections using TLS. Directory Server uses the server's TLS encryption key and the same PIN input methods for attribute encryption.

The server uses randomly generated symmetric cipher keys to encrypt and decrypt attribute data. The server wraps these keys using the public key from the server's TLS certificate. As a consequence, the effective strength of the attribute encryption cannot be higher than the strength of the server's TLS key.

WARNING

Without access to the server's private key, it is not possible to recover the symmetric keys from the wrapped copies. Therefore, back up the server's certificate database regularly. If you lose the key, you will no longer be able to decrypt and encrypt data stored in the database.

7.2. ENABLING ATTRIBUTE ENCRYPTION USING THE COMMAND LINE

This procedure demonstrates how to enable attribute encryption for the **telephoneNumber** attribute in the **userRoot** database using the command line. After you perform the procedure, the server stores existing and new values of this attribute AES-encrypted.

Prerequisites

- You have enabled TLS encryption in Directory Server.

Procedure

1. Export the **userRoot** database:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend export -E
userRoot
```

   The server stores the export in an LDIF file in the **/var/lib/dirsrv/*slapd-instance_name*/ldif/** directory. The **-E** option decrypts attributes that are already encrypted during the export.

2. Enable AES encryption for the **telephoneNumber** attribute:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend attr-encrypt --
add-attr telephoneNumber dc=example,dc=com
```

3. Stop the instance:

```
# dsctl instance_name stop
```

4. Import the LDIF file:

```
# dsctl instance_name ldif2db --encrypted userRoot /var/lib/dirsrv/slapd-
instance_name/ldif/None-userroot-2022_01_24_10_28_27.ldif
```

   The **--encrypted** parameter enables the script to encrypt attributes configured for encryption during the import.

5. Start the instance:

```
# dsctl instance_name start
```

Additional resources

- Enabling TLS-encrypted connections to Directory Server

7.3. ENABLING ATTRIBUTE ENCRYPTION USING THE WEB CONSOLE

This procedure demonstrates how to enable attribute encryption for the **telephoneNumber** attribute in the **userRoot** database using the web console. After you perform the procedure, the server stores existing and new values of this attribute AES-encrypted.

Note that the export and import features in the web console do not support encrypted attributes. Therefore, you must perform these steps on the command line.

Prerequisites

- You have enabled TLS encryption in Directory Server.
- You are logged in to the instance in the web console.

Procedure

1. Export the **userRoot** database:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend export -E
userRoot
```

   The server stores the export in an LDIF file in the **/var/lib/dirsrv/*slapd-instance_name*/ldif/** directory. The **-E** option decrypts attributes that are already encrypted during the export.

2. In the web console, navigate to **Database → Suffixes → *suffix_entry* → Encrypted Attributes**.

3. Enter the attribute to encrypt, and click **Add Attribute**.

4. In the **Actions** menu, select **Stop Instance**.

5. On the command line, import the LDIF file:

```
# dsctl instance_name ldif2db --encrypted userRoot /var/lib/dirsrv/slapd-
instance_name/ldif/None-userroot-2022_01_24_10_28_27.ldif
```

   The **--encrypted** parameter enables the script to encrypt attributes configured for encryption during the import.

6. In the web console, open the **Actions** menu, and select **Start Instance**.

Additional resources

- Enabling TLS-encrypted connections to Directory Server

7.4. GENERAL CONSIDERATIONS AFTER ENABLING ATTRIBUTE ENCRYPTION

Consider the following points after you have enabled encryption for data that is already in the database:

- Unencrypted data can persist in the server's database page pool backing file. To remove this data:

  a. Stop the instance:

```
# dsctl instance_name stop
```

  b. Remove the **/var/lib/dirsrv/slapd-*instance_name*/db/guardian** file:

```
# **rm /var/lib/dirsrv/slapd-instance_name/db/guardian``
```

  c. Start the instance:

```
# dsctl instance_name start
```

- After you enabled have encryption and successfully imported the data, delete the LDIF file with the unencrypted data.

- Directory Server does not encrypt the replication log file. To protect this data, store the replication log on an encrypted disk.

- Data in the server's memory (RAM) is unencrypted and can be temporarily stored in swap partitions. To protect this data, configure encrypted swap space.

IMPORTANT

Even if you delete files that contain unencrypted data, this data can be restored under certain circumstances.

7.5. UPDATING THE TLS CERTIFICATES USED FOR ATTRIBUTE ENCRYPTION

Attribute encryption is based on the TLS certificate of the server. Follow this procedure to prevent that attribute encryption fails after renewing or replacing the TLS certificate.

Prerequisites

- You configured attribute encryption.
- The TLS certificate will expire in the near future.

Procedure

1. Export the **userRoot** database:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend export -E
userRoot
```

The server stores the export in an LDIF file in the **/var/lib/dirsrv/*slapd-instance_name*/ldif/** directory. The **-E** option decrypts attributes that are already encrypted during the export.

2. Create a private key and a certificate signing request (CSR). Skip this step if you want to create them using an external utility.

   - If your host is reachable only by one name, enter:

```
# dsctl instance_name tls generate-server-cert-csr -s
"CN=server.example.com,O=example_organization"
```

   - If your host is reachable by multiple names:

```
# dsctl instance_name tls generate-server-cert-csr -s
"CN=server.example.com,O=example_organization" server.example.com
server.example.net
```

     If you specify the host names as the last parameter, the command adds the Subject Alternative Name (SAN) extension with the **DNS:server.example.com, DNS:server.example.net** entries to the CSR.

The string specified in the **-s *subject*** parameter must be a valid subject name according to RFC 1485. The **CN** field in the subject is required, and you must set it to one of the fully-qualified domain names (FQDN) of the server. The command stores the CSR in the **/etc/dirsrv/slapd-*instance_name*/Server-Cert.csr** file.

3. Submit the CSR to the certificate authority (CA) to get a certificate issued. For further details, see your CA's documentation.

4. Import the server certificate issued by the CA to the NSS database:

   - If you created the private key using the **dsctl tls generate-server-cert-csr** command, enter:

     ```
     # dsconf -D "cn=Directory Manager" ldap://server.example.com security certificate add --file /root/instance_name.crt --name "server-cert" --primary-cert
     ```

     Remember the name of the certificate you set in the **--name** *_certificate_nickname* parameter. You require it in a later step.

   - If you created the private key using an external utility, import the server certificate and the private key:

     ```
     # dsctl instance_name tls import-server-key-cert /root/server.crt /root/server.key
     ```

     Note that the command requires you to specify the path to the server certificate first and then the path to the private key. This method always sets the nickname of the certificate to **Server-Cert**.

5. Import the CA certificate to the NSS database:

   ```
   # dsconf -D "cn=Directory Manager" ldap://server.example.com security ca-certificate add --file /root/ca.crt --name "Example CA"
   ```

6. Set the trust flags of the CA certificate:

   ```
   # dsconf -D "cn=Directory Manager" ldap://server.example.com security ca-certificate set-trust-flags "Example CA" --flags "CT,,"
   ```

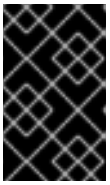
   This configures Directory Server to trust the CA for TLS encryption and certificate-based authentication.

7. Stop the instance:

   ```
   # dsctl instance_name stop
   ```

8. Edit the **/etc/dirsrv/slapd-*instance_name*/dse.ldif** file, and remove the following entries including their attributes:

   - **cn=AES,cn=encrypted attribute keys,cn=*database_name*,cn=ldbm database,cn=plugins,cn=config**
   - **cn=3DES,cn=encrypted attribute keys,cn=*database_name*,cn=ldbm database,cn=plugins,cn=config**



IMPORTANT

Remove the entries for all databases. If any entry that contains the **nsSymmetricKey** attribute is left in the `/etc/dirsrv/slapd-*instance_name*/dse.ldif file, Directory Server will fail to start.

9. Import the LDIF file:

```
# dsctl instance_name ldif2db --encrypted userRoot /var/lib/dirsrv/slapd-instance_name/ldif/None-userroot-2022_01_24_10_28_27.ldif
```

The **--encrypted** parameter enables the script to encrypt attributes configured for encryption during the import.

10. Start the instance:

```
# dsctl instance_name start
```

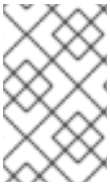
CHAPTER 8. CREATING AND MAINTAINING DATABASE LINKS

When a client application requests data from a database link, it retrieves the data from the remote database and returns it to the client. *Chaining* means that a server contacts other servers on behalf of a client application and then returns the combined results. This chaining process is implemented through a database link.

8.1. CREATING A NEW DATABASE LINK

Basic database link configuration requires the following information:

- **Suffix information**
  The suffix information must correspond to the suffix on the remote server that contains the data. You can create suffix in the directory tree managed by the database link.
- **Bind credentials**
  When the database link binds to a remote server, it impersonates a user, and this specifies the **DN** and the credentials for each database link to use to bind with remote servers.
- **LDAP URL**
  The **LDAP URL** information provides the **LDAP URL** of the remote server to which the database link connects. URL consists of the protocol (**LDAP** or **LADPS**), the host name or IP address (**IPv4** or **IPv6**) for the server, and the port.
- **List of failover servers**
  The list of failover servers is a list of alternative servers for the database link to contact in the case of a failure and is optional.



NOTE

If secure binds are required for simple password authentication, using a secure connection (**TLS** and **STARTTLS** connections or the **SASL** authentication) is recommended when any chaining operations are performed .

8.2. CREATING A NEW DATABASE LINK USING THE COMMAND LINE

You can create a new database link by using the **dsconf chaining link-create** command.

Prerequisites

- You have opened the Directory Server user interface in the web console and selected the instance.

Procedure

1. Create a new database link :

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining link-create --suffix="ou=Customers,dc=example,dc=com" --server-url="ldap://remote_server.example.com:389" --bind-mech=SIMPLE --bind-dn="cn=proxy_user,cn=config" --bind-pw="password" "example_chain_name"
```

By using this command, you create the database link named **example_chain_name** for

ou=Customers,dc=example,dc=com which refers to the **ldap://remote_server.example.com:389** server and uses the specified bind **DN** and **password** to authenticate. You must set bind-mech to **SIMPLE** (**EXTERNAL** for certificate based authentication) or **GSSAPI** for kerberos authentication.

1. Display additional settings that you can set when you create the database link:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining link-create --help
```

NOTE

To grant the **proxy_user** the rights to access data, you must create the proxy ACI entry in the **dc=example,dc=com** suffix on remote server.

Verification

- Display the new database link:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining link-create --
suffix="ou=Customers,dc=example,dc=com" --server-
url="ldap://remote_server.example.com:389" --bind-mech=SIMPLE --bind-
dn="cn=proxy_user,cn=config" --bind-pw="password" "example_chain_name"
```

8.3. CREATING A NEW DATABASE LINK USING THE WEB CONSOLE

You can create a new database link by using the web console.

Prerequisites

- You have opened the Directory Server user interface in the web console and selected the instance.

Procedure

1. Open the **Database** menu.
2. Create a new suffix:
   a. Click **Create Suffix** button.
   b. Enter the **DN** suffix and back end name.
   c. Select **Create The Top Suffix Entry** and click **Create Suffix**.
3. Select the suffix, click **Suffix Tasks** button on the right side, and select **Create Database Link**.
4. Fill the fields with the details about the connection to the remote server.
5. Click **Create Database Link**.

Verification

- Open the **Database** menu and ensure that the new database link appears in this menu.

8.4. MANAGING THE DEFAULT CONFIGURATION FOR NEW DATABASE LINKS

You can manage the default configuration of database links by using the **dsconf chaining** command .

Procedure

1. Display the current default values of the database link:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining config-get-def
```

2. Use the **dsconf chaining config-set-def** command to change the new database links configuration. For example, to set the **response-delay** parameter to **30**, run:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining config-set-def --response-delay 30
```

   The example command sets the default response timeout for all chaining connections. You can overwrite the response timeout for a specific chaining link by using the **dsconf instance chaining link-set** command.

3. To see the list of all parameters you can set, run:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining config-set-def --help
```

CHAPTER 9. SETTINGS REQUIRED FOR CREATING A DATABASE LINK

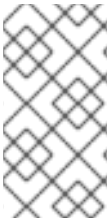
When creating a database link, you must configure the **suffix**, **bind credentials**, **bind mechanisms** and **LDAP URL** settings.

9.1. BIND CREDENTIALS

You can chain request from a client application to remote server by using specific bind credentials. The chained suffix on the remote server must have an ACI that allows proxy authorization to the user. Without bind credentials, a database link binds to the remote server as **anonymous**.

WARNING

When you enable chaining, carefully examine access controls to avoid providing access to restricted areas of a directory. For example, a user that connects by using the database link can see all entries below the branch. To restrict access to the subtree when not all subtrees must be visible to the user, create an additional ACI to avoid any security issues.



NOTE

When a client application creates or modifies entries by using database links, the **creatorsName** and **modifiersName** attributes do not reflect the real creator or modifier of the entries. These attributes contain the name of the administrative user granted proxied authorization rights on the remote data server.

Providing bind credentials involves the following steps on the remote server:

- Creating an administrative user, such as **cn=proxy_user,cn=config**, for the database link.
- Providing proxy access rights for the administrative user created in the previous step on the target subtree chained by using the database link.

For example, the following ACI grants read-only access to the **cn=proxy_admin,cn=config** user to access data contained on the remote server only within the subtree where the ACI is set.

```
aci: (targetattr = "*")(version 3.0; acl "Proxied authorization
    for database links"; allow (proxy) userdn = "ldap:///cn=proxy_admin
    ,cn=config";)
```

NOTE

When a user binds to a database link, the user's identity is sent to the remote server. Access controls are always evaluated on the remote server. To allow the user to modify or write the data to the remote server, set up the correct access controls on the remote server.

Additional resources

- Managing access controls.
- Managing directory attributes.

9.2. LDAP URL

You can identify the remote server that the database link connects with by using **LDAP URL** on a server containing the database link. The URL of the remote server does not specify a suffix and is in the **ldap://host_name:port** form.

The remote server uses the **LDAPS** protocol instead of **LDAP** in the URL and points to the secure port of the server, when you connect the database link to the remote server by using **LDAP** over **TLS**, for example:

```
ldaps://africa.example.com:636/
```

NOTE

You must enable **TLS** on the local Directory Server and the remote Directory Server to be chained over. When the database link and the remote server communicate by using **TLS**, the client application that creates operation request can bind by using the normal port and not necessarily use **TLS** for communication.

Additional resources

- Enabling TLS-encrypted connections to Directory Server

9.3. BIND MECHANISMS

You can connect a local server to a remote server either of the following ways:

- By using a standard **LDAP** port.
- By using a dedicated **LDAPS** port.
- By using the **STARTTLS** connection, which is more secure connection than a standard port.

NOTE

If secure binds are required for simple password authentication, using a secure connection (**TLS** and **STARTTLS** connections or **SASL** authentication) is recommended when you perform any chaining operation.

The local server can use following methods to authenticate to the remote server:

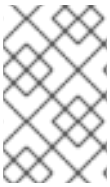
- `EMPTY`s
  When using the **EMPTY** method, the local server performs simple authentication and requires a bind DN and a password if bind mechanism is not set.
- **EXTERNAL**
  When using the **EXTERNAL** method, the local server applies the TLS certificate to authenticate the local server to the remote server. Note that you must either set the local server URL to the

secure URL (**ldaps**) or the **nsUseStartTLS** attribute to **on**. Additionally, you must configure the remote server to map the local server's certificate to its bind identity.

- **DIGEST-MD5**
  When using the **DIGEST-MD5** method, the local server applies the **SASL** authentication with the **DIGEST-MD5** encryption. Similarly to simple authentication, this type of authentication requires the **nsMultiplexorBindDN** and **nsMultiplexorCredentials** attributes to give the bind information.

- **GSSAPI**
  When using the **GSSAPI** method, the local server applies the **Kerberos-based** authentication over the **SASL** authentication.

  You can configure the local server with a **Kerberos** keytab, and the remote server must configure a defined **SASL** mapping for the local server's bind identity.



NOTE

SASL connections can establish over standard connections or **TLS** connections. You can configure local server to chain the **SASL** and password policy components when **SASL** is used.

Additional resources

- Setting up SASL identity mapping

CHAPTER 10. CONFIGURING THE CHAINING POLICY

You can configure Directory Server to chain requests from client applications to Directory Server containing database links. Chaining policy applies to all database links created on Directory Server.

10.1. CHAINING COMPONENT OPERATIONS

A *component* is any functional unit in the server that uses internal operations, for example, a plug-in or function in the front end.

Some components send internal LDAP requests to the server, expecting to access local data only. For such components, you must control the chaining policy so that the components can complete there operations successfully. For example, the certificate verification function. You can chain the LDAP request made by the function to check certificates that implies the remote server is trusted. If the remote server is not trusted, then there is a security problem.

By default, you cannot chain all the internal operations and any component, but the default can be overridden.

Additionally, you must create an **ACI** on the remote server to enable the specified plug-in to perform its operation on the remote server. The **ACI** must exist in the **suffix** assigned to database link.

The following are component names, their potential side-effects of when you allow these components to chain internal operations, and the permissions the components need in the **ACI** on the remote server:

- The **ACI plug-in** component
  The **ACI plug-in** component implements access control. You cannot chain operations used to retrieve and update **ACI** attributes because it not safe to mix the local and the remote attributes. However, you can chain requests used to retrieve user entries by setting the following chaining components attribute:

  ```
  nsActiveChainingComponents: cn=ACI Plugin,cn=plugins,cn=config
  ```

  Permissions: Read, search, and compare.

- The **resource limit** component
  The **resource limits** component sets server limits depending on the user bind DN. If you chain the resource limitation component, you can apply resource limits on the remote users. To chain resource limit component operations, add the following chaining component attribute:

  ```
  nsActiveChainingComponents: cn=resource limits,cn=components,cn=config
  ```

  Permissions: Read, search, and compare.

- The **certificate-based authentication** component
  You can use the **certificate-based authentication** component during the external bind method.This component retrieves user certificates from the database on the remote server. When you allow this component to chain, it enables certificate-based authentication to work with the database link. To chain this component's operations, add the following chaining component attribute:

  ```
  nsActiveChainingComponents: cn=certificate-based
  authentication,cn=components,cn=config
  ```

Permissions: Read, search, and compare.

- The **password policy** component
  The **password policy** component adds **SASL** binds to the remote server. Authenticating with a user name and password is essential for some forms of SASL authentication. When you enable the password policy, it allows the server to verify and implement the specific authentication method requested and to apply the appropriate password policies. To chain this component's operations, add the chaining component attribute:

  ```
  nsActiveChainingComponents: cn=password policy,cn=components,cn=config
  ```

  Permissions: Read, search, and compare.

- The **SASL** component
  The **SASL** component allows SASL to bind to the remote server. To chain this component's operations, add the chaining component attribute:

  ```
  nsActiveChainingComponents: cn=password policy,cn=components,cn=config
  ```

  Permissions: Read, search, and compare.

- The **referential integrity postoperation** component
  The **referential integrity postoperation** component propagates updates made to attributes containing DNs to the entries that contain pointers to the attributes. For example, you can automatically remove an entry from a group when group is deleted. By using the **referential integrity postoperation** plug-in together with the chaining simplifies the management of static group when the group members are remote to the static group definition.

  ```
  nsActiveChainingComponents: cn=referential integrity postoperation,cn=plugins,cn=config
  ```

  Permissions: Read, search, and compare.

- The **attribute Uniqueness** component
  The **attribute Uniqueness** component validates that all the values for a specified attribute are unique. When you chain the plug-in, it confirms that attribute values are unique even when attributes are changed through a database link. To chain this component's operations, add the chaining component attribute:

  ```
  nsActiveChainingComponents: cn=attribute uniqueness,cn=plugins,cn=config
  ```

  Permissions: Read, search, and compare.

- The **roles** component
  The **roles** component chains the roles and roles assignments for the entries in a database. When you chain this component, it maintains the roles even on chained databases. To chain this component's operations, addthe chaining component attribute:

  ```
  nsActiveChainingComponents: cn=roles,cn=components,cn=config
  ```

  Permissions: Read, search, and compare.

NOTE

You cannot chain **Roles** plug-in, **Password policy** component, **Replication** plug-in, and **Referential Integrity** plug-in components. When you enable the **Referential Integrity** plug-in on servers that issue chaining requests, ensure that you analyzed the performance, resource, time, and integrity needs. Not that integrity checks can be time-consuming and draining on memory and CPU.

10.2. CHAINING COMPONENT OPERATIONS USING THE COMMAND LINE

You can add a component allowed to chain by using the command line:

Procedure

1. Specify the components to include in chaining:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining config-set \ --
add-comp="cn=referential integrity postoperation,cn=components,cn=config"
```

2. Restart the instance:

```
# dsctl instance_name restart
```

3. Create an ACI in the suffix on the remote server to which the operation will be chained:

```
# ldapmodify -D "cn=Directory Manager" -W -H 389 remoteserver.example.com -x
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "*")(target="ldap:///ou=customers,ou=People,dc=example,dc=com")
(version 3.0; acl "RefInt Access for chaining"; allow
(read,write,search,compare) userdn = "ldap:///cn=referential
integrity postoperation,cn=plugins,cn=config";)
```

Verification

- Display the components allowed to chain:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining config-set \ --
add-comp="cn=referential integrity postoperation,cn=components,cn=config"
```

10.3. CHAINING COMPONENT OPERATIONS USING THE WEB CONSOLE

You can add a component allowed to chain by using the web console:

Prerequisites

- You have opened the Directory Server user interface in the web console and selected the instance.

Procedure

1. Open the **Database**.
2. In the navigation on the left, select the **Chaining Configuration** entry.
3. Click the **Add** button below the components to **Chain field**.
4. Select the component that you want to chain, and click **Add & Save New Components**.
5. Create **ACI** in the suffix on the remote server to which the operation will be chained:

```
# ldapmodify -D "cn=Directory Manager" -W -H 389 remoteserver.example.com -x
dn: ou=People,dc=example,dc=com
changetype: modify
add: aci
aci: (targetattr = "*")(target="ldap:///ou=customers,ou=People,dc=example,dc=com")
(version 3.0; acl "RefInt Access for chaining"; allow
(read,write,search,compare) userdn = "ldap:///cn=referential
integrity postoperation,cn=plugins,cn=config";)
```

Verification

- Selected component should be chained .

CHAPTER 11. CHAINING LDAP CONTROLS

LDAP operations can contains some data (named control and specified in the ldap protocol) that change their behavior. You can specify which LDAP controls to forward to the remote server.

11.1. ABOUT CHAINING LDAP CONTROLS

The database link forwards requests containing the following controls to the remote server for chaining LDAP controls:

- The **Virtual List View (VLV)** control provides lists of certain entries.
- The **Server-side sorting** control categorizes entries according to their attribute values, usually by using a specific matching rule.
- The **Dereferencing** control pulls specified attribute information from the referenced entry and returns this information with the rest of the search results.
- The **Managed DSA** controls returns smart referrals as entries, rather than following these referrals. Therefore, a smart referral itself can be changed or deleted.
- The **Loop detection** control tracks how many times a server chains with another server. When the count reaches the configured number, **Loop detection** detects a loop and notifies the client application.

NOTE

Database links cannot support **Server-side sorting** and **VLV** controls when a client application makes a request to multiple databases.

The following are some of the **LDAP** controls that are allowed to chain and their object identifiers (OID):

Control Name	OID
Virtual list view (VLV)	2.16.840.1.113730.3.4.9
Server-side sorting	1.2.840.113556.1.4.473
Managed DSA	2.16.840.1.113730.3.4.2
Loop detection	1.3.6.1.4.1.1466.29539.12
Dereferencing searches	1.3.6.1.4.1.4203.666.5.16

11.2. CHAINING LDAP CONTROLS USING THE COMMAND LINE

You can chain **LDAP** controls by using **dsconf chaining config-set --add-control** in the command line.

Procedure

1. Chain **LDAP** controls:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com chaining \ config-set --
add-control="2.16.840.1.113730.3.4.9"
```

Add the object identifier (OID) of the custom control if clients of Directory Server create their own controls and chain there operations to remote servers.

11.3. CHAINING LDAP CONTROLS USING THE WEB CONSOLE

You can chain **LDAP** controls by using the web console.

Prerequisites

- You have opened the Directory Server user interface in the web console and selected the instance.

Procedure

1. Open the **Database** menu.
2. Select the **Chaining Configuration** entry.
3. Click **Add** button below the **Forwarded LDAP Controls** field.
4. Select the LDAP control and click **Add & Save New Controls** button.
   Add the object identifier (OID) of the custom control if clients of Directory Server create their own controls and chain there operations to remote servers.
5. Click **Save** button.

Verification

- Click the **Database** menu and ensure that the selected **LDAP** control is chained.

CHAPTER 12. DATABASE LINKS AND ACCESS CONTROL EVALUATION

When a user binds to a server containing a database link, the database link sends the user's identity to the remote server. You can evaluate access control on the remote server.

You can evaluate the **LDAP** operation on the remote server by using the original identity of the client application passed by using the proxied authorization control.

NOTE

You must have the correct access controls on the subtree present on the remote server for the operations to succeed on the remote server.

You can add usual access controls to the remote server with the following restrictions:

- You cannot use all types of access control. For example, role-based or filter-based ACIs need access to the user entry, because the data is accessed through database links.
- Remote server views the client application in the same IP address and DNS domain as the database link. Because the original domain of the client is lost during chaining, all access controls based on the IP address or DNS domain of the client cannot work.

NOTE

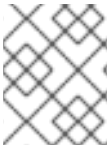
Directory Server supports both **IPv4** and **IPv6** IP addresses.

The following restrictions apply to the ACIs used with database links:

- You must locate ACIs with any groups they use. For the dynamic groups, all users in the group are located with the ACI and the group. For the static group, user links to remote server.
- You must locate ACIs with any role definitions they use and with any users who intend to use these roles.
- ACIs that link to values of a user's entry must work when the user is remote.

Though evaluation of access controls is always done on the remote server, access controls can also be evaluated on both the server containing the database link and the remote server. This poses the following several limitations:

- When you evaluate the access control, for example, on the server containing the database link and when the entry is located on a remote server, the contents of user entries are not necessarily available.



NOTE

For performance reasons, clients cannot perform remote inquiries and evaluate access controls.

- When you perform modify operation, the database link does not have access to the full entry stored on the remote server and necessarily does not have access to the entries being modified by the client application.

- When you perform delete operation, the database link is only aware of the entry's **DN**. If an access control specifies a particular attribute, then delete operation must fail when conducted through a database link.

NOTE

By default, evaluation of access controls on the server containing the database link is not allowed. You can override this default setting by using the **nsCheckLocalACI** attribute in the **cn=database_link**, **cn=chaining database**, **cn=plugins**, and **cn=config** entry. However, evaluating access controls on the server containing the database link is not recommended except for cascading chaining.