Red Hat Directory Server 12

Configuring directory databases

Configuring and managing Red Hat Directory Server databases

Last Updated: 2022-05-18

Red Hat Directory Server 12 Configuring directory databases

Configuring and managing Red Hat Directory Server databases

Legal Notice

Abstract

This documentation describes tasks related to the databases in Directory Server. For example, this includes creating suffixes and databases, as well as configuring chaining, database links, and referrals.

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 simple comments on specific passages:
  1. Make sure you are viewing the documentation in the *Multi-page HTML* format. In addition, ensure you see the **Feedback** button in the upper right corner of the document.
  2. Use your mouse cursor to highlight the part of text that you want to comment on.
  3. Click the **Add Feedback** pop-up that appears below the highlighted text.
  4. Follow the displayed instructions.
- For submitting more complex feedback, create a Bugzilla ticket:
  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

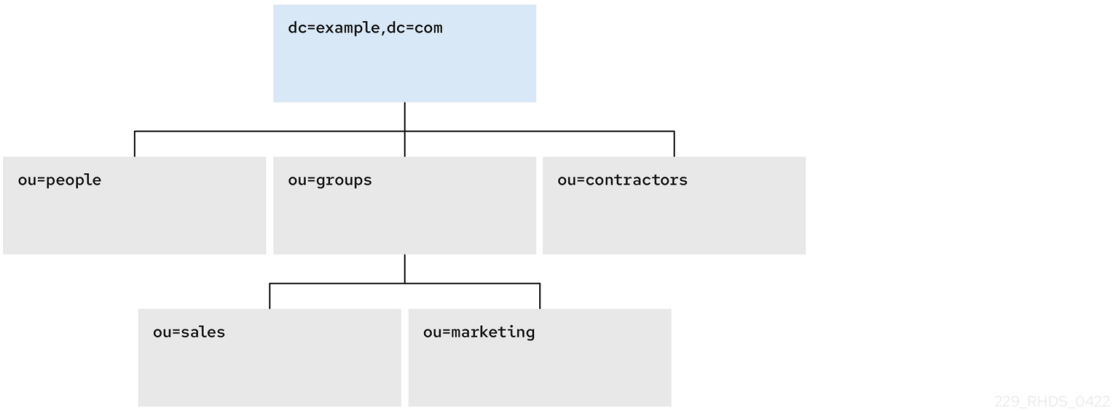
You can design distributed data storage logic in Directory Server by dividing data in an instance into several databases. You can use suffixes of directory trees as the method of data division.

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.

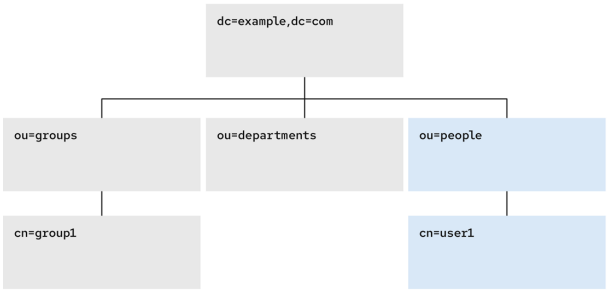
This way you can have multiple databases in a single instance of the server. You are not confined to a single database.

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:

- Database maintenance (import/export/indexing) is easier.
- Sub-suffixes can be stored on separate disks, which helps with disk space concerns.

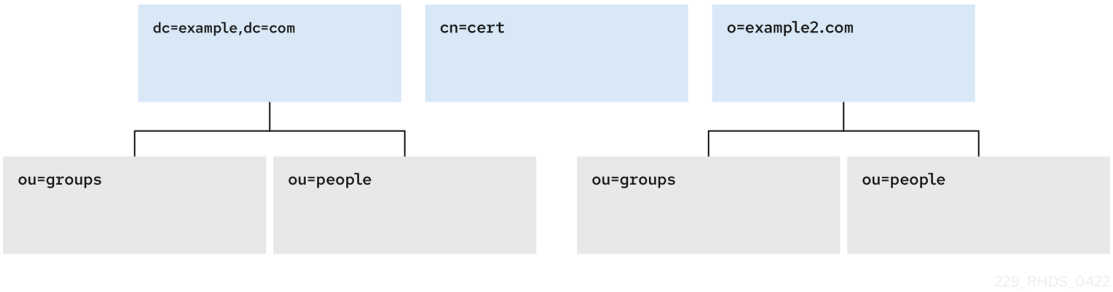
Disadvantages of using sub-suffixes:

- Clients cannot query the root suffix and have sub-suffix entries returned. Clients must begin a search at the sub-suffix level to retrieve entries from that sub-suffix.
- 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

This procedure instructs you how to create a sub-suffix of a directory tree on 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 and associate it with a new database using the **--be-name** argument and an existing parent suffix using 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"
```

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.

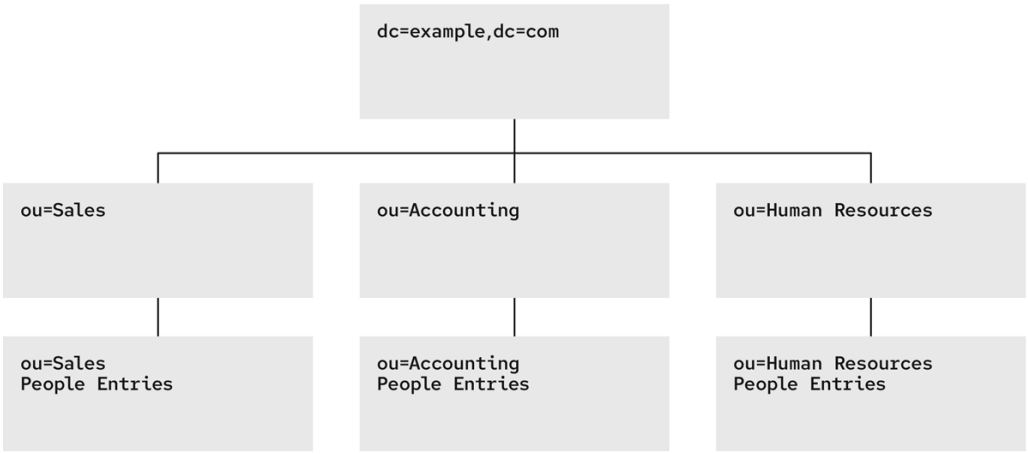
NOTE

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.

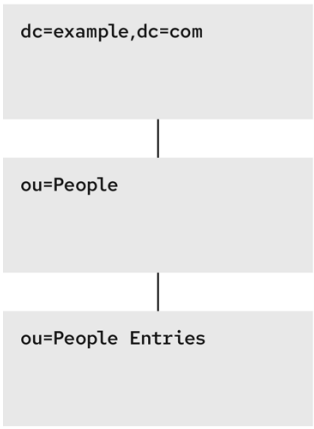
Figure 2.1. Standard hierarchy DIT based on functional units



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.

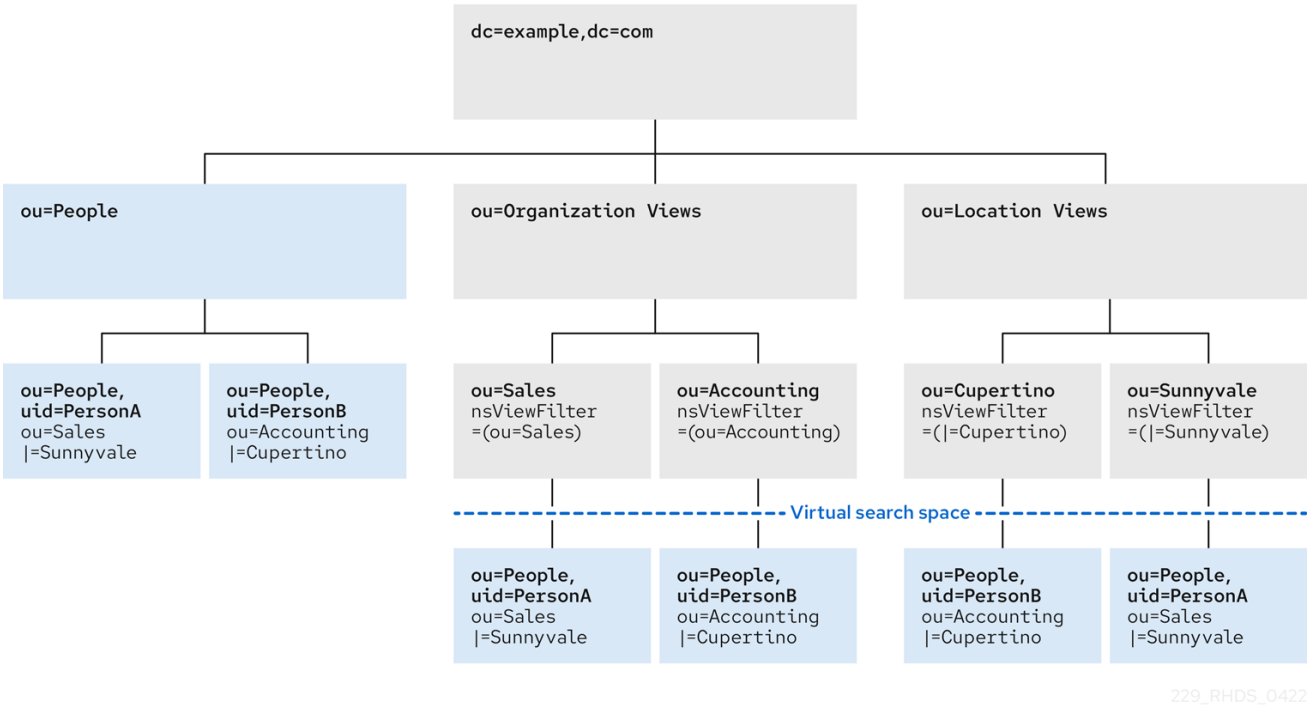
Figure 2.2. Standard flat DIT based on resource types



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,,"
   ```

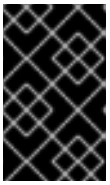
   This configures Directory Server to trust the CA for TLS encryption and certificate-based authentication.

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
```