Red Hat Directory Server 12

Managing indexes

Improving search performance by optimizing indexes

Last Updated: 2024-07-09

Red Hat Directory Server 12 Managing indexes

Improving search performance by optimizing indexes

Legal Notice

Abstract

Indexing makes searching for and retrieving information faster by classifying and organizing attributes or values. You can request a contiguous subset of a large search result by using virtual list view control.

Table of Contents

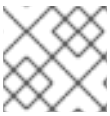
PROVIDING FEEDBACK ON RED HAT DIRECTORY SERVER

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

- For submitting feedback on the Red Hat Directory Server documentation through Jira (account required):
  1. Go to the Red Hat Issue Tracker.
  2. Enter a descriptive title in the **Summary** field.
  3. Enter your suggestion for improvement in the **Description** field. Include links to the relevant parts of the documentation.
  4. Click **Create** at the bottom of the dialogue.
- For submitting feedback on the Red Hat Directory Server product through Jira (account required):
  1. Go to the Red Hat Issue Tracker.
  2. On the **Create Issue** page, click **Next**.
  3. Fill in the **Summary** field.
  4. Select the component in the **Component** field.
  5. Fill in the **Description** field including:
     a. The version number of the selected component.
     b. Steps to reproduce the problem or your suggestion for improvement.
  6. Click **Create**.

CHAPTER 1. DEFINING A DEFAULT INDEX THAT APPLIES TO ALL NEWLY CREATED DATABASES

The default index in Directory Server defines a set of attributes to be indexed. When you create a new database, Directory Server copies the default index attributes from **cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config** entry to the database-specific **cn=index,cn=*database_name*,cn=ldbm database,cn=plugins,cn=config** entry.



NOTE

Directory Server does not apply changes in the default index to existing databases.

1.1. THE DIFFERENT INDEX TYPES

Directory Server stores the indexes of each indexed attribute in a separate database file in the instance's database directory. For example, the indexes of the **sn** attribute are stored in the **/var/lib/dirsrv/slapd-*instance_name*/db/*database_name*/sn.db** file. Each index file can contain multiple index types if Directory Server maintains different indexes for an attribute.

Directory Server supports the following index types:

- The presence index (**pres**) is a list of the entries that contain a particular attribute. For example, use this type when clients frequently perform searches, such as **attribute=mail**.
- The equality index (**eq**) improves searches for entries containing a specific attribute value. For example, an equality index on the **cn** attribute enables faster searches for **cn=*first_name last_name***.
- The approximate index (**approx**) enables efficient approximate or sounds-like searches. For example, searches for **cn~=*first_name last_name***, **cn~=*first_name***, or **cn~=*first_nam*** (note the misspelling) would return an entry **cn=*first_name X last_name***. Note that the metaphone phonetic algorithm in Directory Server supports only US-ASCII letters. Therefore, use approximate indexing only with English values.
- The substring index (**sub**) is a costly index to maintain, but it enables efficient searching against substrings within entries. Substring indexes are limited to a minimum of three characters for each entry. For example, searches for **telephoneNumber=*555**** return all entries in the directory with a value that contains **555** in the **telephoneNumber** attribute.
- International index speeds up searches for information in international directories. The process for creating an international index is similar to the process for creating regular indexes, except that it applies a matching rule by associating an object identifier (OID) with the attributes to be indexed.

1.2. BALANCING THE BENEFITS OF INDEXING

Before you create new indexes, balance the benefits of maintaining indexes against the costs:

- Approximate indexes are not efficient for attributes commonly containing numbers, such as phone numbers.
- Substring indexes do not work for binary attributes.
- Avoid equality indexes on attributes that contain big values, such as an image.

- Maintaining indexes for attributes that are not commonly used in searches increases the overhead without improving the search performance.
- Attributes that are not indexed can still be used in search requests, although the search performance can be degraded significantly, depending on the type of search.

Indexes can become very time-consuming. For example, if Directory Server receives an add operation, the server examines the indexing attributes to determine whether an index is maintained for the attribute values. If the created attribute values are indexed, Directory Server adds the new attribute values to the index, and then the actual attribute values are created in the entry.

Example 1.1. Indexing steps Directory Server performs when a user adds an entry

Assume that Directory Server maintains the following indexes:

- Equality, approximate, and substring indexes for the **cn** and **sn** attributes.
- Equality and substring indexes for the **telephoneNumber** attribute.
- Substring indexes for the **description** attribute.

For example, a user adds the following entry:

```
dn: cn=John Doe,ou=People,dc=example,dc=com
objectclass: top
objectClass: person
objectClass: orgperson
objectClass: inetorgperson
cn: John Doe
cn: John
sn: Doe
ou: Manufacturing
ou: people
telephoneNumber: 408 555 8834
description: Manufacturing lead
```

When the user adds the entry, Directory Server performs the following steps:

1. Create the **cn** equality index entry for **John** and **John Doe**.
2. Create the **cn** approximate index entries for **John** and **John Doe**.
3. Create the **cn** substring index entries for **John** and **John Doe**.
4. Create the **sn** equality index entry for **Doe**.
5. Create the **sn** approximate index entry for **Doe**.
6. Create the **sn** substring index entry for **Doe**.
7. Create the **telephoneNumber** equality index entry for **408 555 8834**.
8. Create the **telephoneNumber** substring index entry for **408 555 8834**.
9. Create the **description** substring index entry for **Manufacturing lead**.

This example illustrates that the number of actions required to create and maintain databases for a large directory can be very resource-intensive.

IMPORTANT

Do not define a substring index for membership attributes (for example, **member**,**uniquemember**) because it can impact Directory Server performance. When adding or removing members, for example,**uniquemember** to a group with many members, the computation of the **uniquemember** substring index requires to evaluating all **uniquemember** values and not only added or removed values.

1.3. DEFAULT INDEX ATTRIBUTES

Directory Server stores the default index attributes in the **cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config** entry. To display them, including their index types, enter:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -b "cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config" -s one -o ldif-wrap=no
```

Table 1.1. Directory Server default index attributes

aci	**cn**	**entryUSN**
entryUUID	**givenName**	**mail**
mailAlternateAddress	**mailHost**	**member**
memberOf	**nsUniqueId**	**nsCertSubjectDN**
nsTombstoneCSN	**ntUniqueId**	**ntUserDomainId**
numSubordinates	**objectClass**	**owner**
parentId	**seeAlso**	**sn**
targetUniqueId	**telephoneNumber**	**uid**
uniqueMember		

WARNING

Removing the attributes listed in the table (system indexes) from the index of databases can significantly affect the Directory Server performance.

1.4. MAINTAINING THE DEFAULT INDEX

Directory Server stores the default index attributes in the **cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config** entry. Note that you can only maintain the default index attributes using LDIF statements.

Procedure

- For example, to add the **roomNumber** attribute to the default index with the index types **eq** and **sub**, enter:

```
# ldapadd -D "cn=Directory Manager" -W -H ldap://server.example.com -x

dn: cn=roomNumber,cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config
objectClass: nsIndex
objectClass: top
cn: roomNumber
nsSystemIndex: false
nsIndexType: eq
nsIndexType: sub
```

  Explanation of the LDIF statement:

  - **objectClass: nsIndex**: Defines that this entry is an index entry.
  - **objectClass: top**: This object class is additionally required in index entries.
  - **cn**: Sets the name of the attribute to index.
  - **nsSystemIndex**: Indicates whether or not the index is essential to Directory Server operations.
  - **nsIndexType**: This multi-value attribute specifies the index types.

- For example, to add the **pres** index type to the default index attributes of the **roomNumber** attribute, enter:

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

dn: cn=roomNumber,cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config
changetype: modify
add: nsIndexType
nsIndexType: pres
```

- For example, to remove the **pres** index type from the default index attributes of the **roomNumber** attribute, enter:

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

dn: cn=roomNumber,cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config
changetype: modify
delete: nsIndexType
nsIndexType: pres
```

- For example, to remove the **roomNumber** attribute from the default index, enter:

```
# ldapdelete -D "cn=Directory Manager" -W -H ldap://server.example.com -x
cn=roomNumber,cn=default indexes,cn=config,cn=ldbm
database,cn=plugins,cn=config
```

Verification

- List the default index attributes to verify your changes:

```
# ldapsearch -H ldap://server.example.com:389 -D "cn=Directory Manager" -W -b
"cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config" -x -s one -o
ldif-wrap=no
```

CHAPTER 2. MAINTAINING THE INDEXES OF A SPECIFIC DATABASE

Each database in Directory Server has its own index. You can create, update, and delete indexes using the **dsconf** utility or the web console.

2.1. THE DIFFERENT INDEX TYPES

Directory Server stores the indexes of each indexed attribute in a separate database file in the instance's database directory. For example, the indexes of the **sn** attribute are stored in the **/var/lib/dirsrv/slapd-*instance_name*/db/*database_name*/sn.db** file. Each index file can contain multiple index types if Directory Server maintains different indexes for an attribute.

Directory Server supports the following index types:

- The presence index (**pres**) is a list of the entries that contain a particular attribute. For example, use this type when clients frequently perform searches, such as **attribute=mail**.
- The equality index (**eq**) improves searches for entries containing a specific attribute value. For example, an equality index on the **cn** attribute enables faster searches for **cn=*first_name last_name***.
- The approximate index (**approx**) enables efficient approximate or sounds-like searches. For example, searches for **cn~=*first_name last_name***, **cn~=*first_name***, or **cn~=*first_nam*** (note the misspelling) would return an entry **cn=*first_name X last_name***. Note that the metaphone phonetic algorithm in Directory Server supports only US-ASCII letters. Therefore, use approximate indexing only with English values.
- The substring index (**sub**) is a costly index to maintain, but it enables efficient searching against substrings within entries. Substring indexes are limited to a minimum of three characters for each entry. For example, searches for **telephoneNumber=**555**** return all entries in the directory with a value that contains **555** in the **telephoneNumber** attribute.
- International index speeds up searches for information in international directories. The process for creating an international index is similar to the process for creating regular indexes, except that it applies a matching rule by associating an object identifier (OID) with the attributes to be indexed.

2.2. BALANCING THE BENEFITS OF INDEXING

Before you create new indexes, balance the benefits of maintaining indexes against the costs:

- Approximate indexes are not efficient for attributes commonly containing numbers, such as phone numbers.
- Substring indexes do not work for binary attributes.
- Avoid equality indexes on attributes that contain big values, such as an image.
- Maintaining indexes for attributes that are not commonly used in searches increases the overhead without improving the search performance.
- Attributes that are not indexed can still be used in search requests, although the search performance can be degraded significantly, depending on the type of search.

Indexes can become very time-consuming. For example, if Directory Server receives an add operation, the server examines the indexing attributes to determine whether an index is maintained for the attribute values. If the created attribute values are indexed, Directory Server adds the new attribute values to the index, and then the actual attribute values are created in the entry.

Example 2.1. Indexing steps Directory Server performs when a user adds an entry

Assume that Directory Server maintains the following indexes:

- Equality, approximate, and substring indexes for the **cn** and **sn** attributes.
- Equality and substring indexes for the **telephoneNumber** attribute.
- Substring indexes for the **description** attribute.

For example, a user adds the following entry:

```
dn: cn=John Doe,ou=People,dc=example,dc=com
objectclass: top
objectClass: person
objectClass: orgperson
objectClass: inetorgperson
cn: John Doe
cn: John
sn: Doe
ou: Manufacturing
ou: people
telephoneNumber: 408 555 8834
description: Manufacturing lead
```

When the user adds the entry, Directory Server performs the following steps:

1. Create the **cn** equality index entry for **John** and **John Doe**.
2. Create the **cn** approximate index entries for **John** and **John Doe**.
3. Create the **cn** substring index entries for **John** and **John Doe**.
4. Create the **sn** equality index entry for **Doe**.
5. Create the **sn** approximate index entry for **Doe**.
6. Create the **sn** substring index entry for **Doe**.
7. Create the **telephoneNumber** equality index entry for **408 555 8834**.
8. Create the **telephoneNumber** substring index entry for **408 555 8834**.
9. Create the **description** substring index entry for **Manufacturing lead**.

This example illustrates that the number of actions required to create and maintain databases for a large directory can be very resource-intensive.

IMPORTANT

Do not define a substring index for membership attributes (for example, **member**,**uniquemember**) because it can impact Directory Server performance. When adding or removing members, for example,**uniquemember** to a group with many members, the computation of the **uniquemember** substring index requires to evaluating all **uniquemember** values and not only added or removed values.

2.3. DEFAULT INDEX ATTRIBUTES

Directory Server stores the default index attributes in the **cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config** entry. To display them, including their index types, enter:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -b "cn=default indexes,cn=config,cn=ldbm database,cn=plugins,cn=config" -s one -o ldif-wrap=no
```

Table 2.1. Directory Server default index attributes

aci	**cn**	**entryUSN**
entryUUID	**givenName**	**mail**
mailAlternateAddress	**mailHost**	**member**
memberOf	**nsUniqueId**	**nsCertSubjectDN**
nsTombstoneCSN	**ntUniqueId**	**ntUserDomainId**
numSubordinates	**objectClass**	**owner**
parentId	**seeAlso**	**sn**
targetUniqueId	**telephoneNumber**	**uid**
uniqueMember		

WARNING

Removing the attributes listed in the table (system indexes) from the index of databases can significantly affect the Directory Server performance.

2.4. MAINTAINING THE INDEXES OF A SPECIFIC DATABASE USING THE COMMAND LINE

You can use the **dsconf** utility to maintain index settings using the command line.

Procedure

- For example, to add the **roomNumber** attribute to the index of the **userRoot** database with the index types **eq** and **sub**, enter:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend index add --attr roomNumber --index-type eq --index-type sub --reindex userRoot
```

  The **--reindex** option causes that Directory Server automatically re-indexes the database.

- For example, to add the **pres** index type to the index settings of the **roomNumber** attribute in the **userRoot** database, enter:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend index set --attr roomNumber --add-type pres userRoot
```

- For example, to remove the **pres** index type from the index settings of the **roomNumber** attribute in the **userRoot** database, enter:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend index set --attr roomNumber --del-type pres userRoot
```

- For example, to remove the **roomNumber** attribute from the index in the **userRoot** database, enter:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend index delete --attr roomNumber userRoot
```

Verification

- List the index settings of the **userRoot** database:

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

2.5. RECREATING AN INDEX WHILE THE INSTANCE OFFLINE

You can use the **dsctl db2index** utility for reindexing the whole database while the instance is offline.

Prerequisites

- You created an indexing entry or added additional index types to the existing **userRoot** database.

Procedure

1. Shut down the instance:

```
# dsctl instance_name stop
```

2. Recreate the index:

   a. For all indexes in the database, run:

```
# dsctl instance_name db2index

[23/Feb/2023:05:38:28.034826108 -0500] - INFO - check_and_set_import_cache -
pagesize: 4096, available bytes 1384095744, process usage 27467776
[23/Feb/2023:05:38:28.037952026 -0500] - INFO - check_and_set_import_cache -
Import allocates 540662KB import cache.
[23/Feb/2023:05:38:28.055104135 -0500] - INFO - bdb_db2index - userroot: Indexing
attribute: aci
...
[23/Feb/2023:05:38:28.134350191 -0500] - INFO - bdb_db2index - userroot: Finished
indexing.
[23/Feb/2023:05:38:28.151907852 -0500] - INFO - bdb_pre_close - All database threads
now stopped
db2index successful
```

b. For specific attribute indexes, run:

```
# dsctl instance_name db2index userRoot --attr aci cn givenname
```

The following command recreates indexes for **aci**, **cn**, and **givenname** attributes.

c. For more information regarding **dsctl** (offline) command, run:

```
# dsctl instance_name db2index --help
```

3. Start the instance:

```
# dsctl instance_name start
```

Verification

- List the index settings of the **userRoot** database:

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

2.6. MAINTAINING THE INDEXES OF A SPECIFIC DATABASE USING THE WEB CONSOLE

You can use the web console to maintain index settings in Directory Server.

Prerequisites

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

Procedure

- Navigate to **Database** → **Suffixes** → *suffix_name* → **Indexes** → **Database Indexes**.
  - To add an attribute to the index:
    - Click **Add Index**.

        - Enter the attribute name to the **Select An Attribute** field.
        - Select the index types.
        - Select **Index attribute after creation**.
        - Click **Create Index**.
    - To update the index settings of an attribute:
        - Click the overflow menu next to the attribute, and select **Edit Index**.
        - Update the index settings to your needs.
        - Select **Index attribute after creation**.
        - Click **Save Index**.
    - To delete an attribute from the index:
        - Click the overflow menu next to the attribute, and select **Delete Index**.
        - Select **Yes, I am sure**, and click **Delete**.
        - In the **Suffix Tasks** menu, select **Reindex Suffix**.

Verification

- Navigate to **Database → Suffixes →** *suffix_name* **→ Indexes → Database Indexes**, and verify that the index settings reflect the changes you made.

CHAPTER 3. CHANGING THE SEARCH KEY LENGTH IN A SUBSTRING INDEX

By default, the length of the search key for substring indexes must be at least **three** characters. For example, Directory Server will add the string **abc** as a search key to an index while **ab*** will not. However, to improve the search performance, particularly for searches with many wildcard characters, you can shorten the search key length. This increases the number of search keys in the index.

Directory Server has three attributes that change the minimum number of characters required for a search key:

- **nsSubStrBegin**: Sets the minimum number of characters for the beginning of a search key, before the wildcard character. For example:

```
abc*
```

- **nsSubStrMiddle**: Sets the minimum number of characters in the search key between wildcard characters. For example:

```
*abc*
```

- **nsSubStrEnd**: Sets the number of characters for the end of a search key, after the wildcard character. For example:

```
*xyz
```

3.1. CHANGING THE SEARCH KEY LENGTH IN A SUBSTRING INDEX USING THE COMMAND LINE

You can improve search speeds by setting a new search key length for an attribute index.

Procedure

1. To set new search key length, add the **extensibleObject** object class and then add the **nsSubStrBegin**, **nsSubStrEnd**, or **nsSubStrMiddle** attributes to the entry. For example:

```
# ldapmodify -D "cn=Directory Manager" -W -p 389 -h server.example.com -x

dn: attribute_name,cn=index,cn=database_name,cn=ldbm database,cn=plugins,cn=config
changetype: modify
add: objectclass
objectclass: extensibleObject
-
add: nsSubStrBegin
nsSubStrBegin: 2
-
add: nsSubStrMiddle
nsSubStrMiddle: 2
-
add: nsSubStrEnd
nsSubStrEnd: 2
```

2. Recreate the index to apply new setting. For example, while the Directory Server instance is running, use the following command to recreate the index for the specified attribute:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend index reindex --attr
attribute_name database_name
```

Verification

- Select the attribute for which you want to change the search key length, for example, **cn**.
- Dump the **cn** index:

```
dbscan -D bdb -f /var/lib/dirsrv/slapd-instance/db/database/cn.db > /tmp/default_len
```

- Configure the new search key length as described in section Changing the search key length in a substring index using the command line.
- Stop the instance to synchronize the database on the disk:

```
# dsctl instance_name stop
```

- Dump the **cn** index:

```
dbscan -D bdb -f /var/lib/dirsrv/slapd-instance/db/database/cn.db > /tmp/len_2
```

- Compare **len_2** and **default_len** files:

```
diff /tmp/len_2 /tmp/default_len
```

CHAPTER 4. USING VIRTUAL LIST VIEW CONTROL TO REQUEST A CONTIGUOUS SUBSET OF A LARGE SEARCH RESULT

Directory Server supports the LDAP virtual list view control. This control enables an LDAP client to request a contiguous subset of a large search result.

For example, you have stored an address book with 100.000 entries in Directory Server. By default, a query for all entries returns all entries at once. This is a resource and time-consuming operation, and clients often do not require the whole data set because, if the user scrolls through the results, only a partial set is visible.

However, if the client uses the VLV control, the server only returns a subset and, for example, if the user scrolls in the client application, the server returns more entries. This reduces the load on the server, and the client does not need to store and process all data at once.

VLV also improves the performance of server-sorted searches when all search parameters are fixed. Directory Server pre-computes the search results within the VLV index. Therefore, the VLV index is much more efficient than retrieving the results and sorting them afterwards.

In Directory Server, the VLV control is always available. However, if you use it in a large directory, a VLV index, also called browsing index, can significantly improve the speed.

Directory Server does not maintain VLV indexes for attributes, such as for standard indexes. The server generates VLV indexes dynamically based on attributes set in entries and the location of those entries in the directory tree. Unlike standard entries, VLV entries are special entries in the database.

4.1. HOW THE VLV CONTROL WORKS IN LDAPSEARCH COMMANDS

Typically, you use the virtual list view (VLV) feature in LDAP client applications. However, for example for testing purposes, you can use the **ldapsearch** utility to request only partial results.

To use the VLV feature in **ldapsearch** commands, specify the **-E** option for both the **sss** (server-side sorting) and **vlv** search extensions:

```
# ldapsearch ... -E 'sss=attribute_list' -E 'vlv=query_options'
```

The **sss** search extension has the following syntax:

```
[!]sss=[-]<attr[:OID]>[/[-]<attr[:OID]>...]
```

The **vlv** search extension has the following syntax:

```
[!]vlv=<before>/<after>(/<offset>/<count>|:<value>)
```

- **before** sets the number of entries returned before the targeted one.
- **after** sets the number of entries returned after the targeted one.
- **index**, **count**, and **value** help to determine the target entry. If you set **value**, the target entry is the first one having its first sorting attribute starting with the value. Otherwise, you set **count** to **0**, and the target entry is determined by the **index** value (starting from 1). If the **count** value is higher than **0**, the target entry is determined by the ratio **index * number of entries** / **count**.

Example 4.1. Output of an ldapsearch command with VLV search extension

The following command searches in **ou=People,dc=example,dc=com**. The server then sorts the results by the **cn** attribute and returns the **uid** attributes of the 70th entry together with one entry before and two entries after the offset.

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -b
"ou=People,dc=example,dc=com" -s one -x -E 'sss=cn' -E 'vlv=1/2/70/0' uid
# user069, People, example.com
dn: uid=user069,ou=People,dc=example,dc=com
uid: user069

# user070, People, example.com
dn: uid=user070,ou=People,dc=example,dc=com
uid: user070

# user071, People, example.com
dn: uid=user071,ou=People,dc=example,dc=com
uid: user071

# user072, People, example.com
dn: uid=user072,ou=People,dc=example,dc=com
uid: user072

# search result
search: 2
result: 0 Success
control: 1.2.840.113556.1.4.474 false MIQAAAADCgEA
sortResult: (0) Success
control: 2.16.840.1.113730.3.4.10 false MIQAAAALAgFGAgMAnaQKAQA=
vlvResult: pos=70 count=40356 context= (0) Success

# numResponses: 5
# numEntries: 4
Press [before/after(/offset/count|:value)] Enter for the next window.
```

Additional resources

The **-E** parameter description in the **ldapsearch(1)** man page.

4.2. ENABLING UNAUTHENTICATED USERS TO USE THE VLV CONTROL

By default, the access control instruction (ACI) in the **oid=2.16.840.1.113730.3.4.9,cn=features,cn=config** entry enables only authenticated users to use the VLV control. To enable also non-authenticated users to use the VLV control, update the ACI by changing **userdn = "ldap:///all"** to **userdn = "ldap:///anyone"**

Procedure

- Update the ACI in **oid=2.16.840.1.113730.3.4.9,cn=features,cn=config**:

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

```
dn: oid=2.16.840.1.113730.3.4.9,cn=features,cn=config
changetype: modify
replace: aci
aci: (targetattr != "aci")(version 3.0; acl "VLV Request Control"; allow( read, search,
compare, proxy ) userdn = "ldap:///anyone";)
```

Verification

- Perform a query with VLV control not specify a bind user:

```
# ldapsearch -H ldap://server.example.com -b "ou=People,dc=example,dc=com" -s one
-x -E 'sss=cn' -E 'vlv=1/2/70/0' uid
```

This command requires that the server allows anonymous binds.

If the command succeeds but returns no entries, run the query again with a bind user to ensure that the query works when using authentication.

Additional resources

- Disabling anonymous binds

4.3. CREATING A VLV INDEX USING THE COMMAND LINE TO IMPROVE THE SPEED OF VLV QUERIES

Follow this procedure to create a virtual list view (VLV) index, also called browsing index, for entries in **ou=People,dc=example,dc=com** that contain a **mail** attribute and have the **objectClass** attribute set to **person**.

Prerequisites

- Your client applications use the VLV control.
- Client applications require to query a contiguous subset of a large search result.
- The directory contains a large number of entries.

Procedure

1. Create the VLV search entry:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend vlv-index
add-search --name "VLV People" --search-base "ou=People,dc=example,dc=com" --
search-filter "(&(objectClass=person)(mail=*))" --search-scope 2 userRoot
```

This command uses the following options:

- **--name** sets the name of the search entry. This can be any name.
- **--search-base** sets the base DN for the VLV index. Directory Server creates the VLV index on this entry.

- **--search-scope** sets the scope of the search to run for entries in the VLV index. You can set this option to **0** (base search), **1** (one-level search), or **2** (subtree search).
- **--search-filter** sets the filter Directory Server applies when it creates the VLV index. Only entries that match this filter become part of the index.
- **userRoot** is the name of the database in which to create the entry.

2. Create the index entry:

```
# dsconf -D "cn=Directory Manager" ldap://server.example.com backend vlv-index
add-index --index-name "VLV People - cn sn" --parent-name "VLV People" --sort "cn
sn" --index-it userRoot
```

This command uses the following options:

- **--index-name** sets the name of the index entry. This can be any name.
- **--parent-name** sets the name of the VLV search entry and must match the name you set in the previous step.
- **--sort** sets the attribute names and their sort order. Separate the attributes by space.
- **--index-it** causes that Directory Server automatically starts an index task after the entry was created.
- **userRoot** is the name of the database in which to create the entry.

Verification

1. Verify the successful creation of the VLV index in the **/var/log/dirsrv/slapd-*instance_name*/errors** file:

```
[26/Nov/2021:11:32:59.001988040 +0100] - INFO - bdb_db2index - userroot: Indexing VLV:
VLV People - cn sn
[26/Nov/2021:11:32:59.507092414 +0100] - INFO - bdb_db2index - userroot: Indexed 1000
entries (2%).
...
[26/Nov/2021:11:33:21.450916820 +0100] - INFO - bdb_db2index - userroot: Indexed 40000
entries (98%).
[26/Nov/2021:11:33:21.671564324 +0100] - INFO - bdb_db2index - userroot: Finished
indexing.
```

2. Use the VLV control in an **ldapsearch** command to query only specific records from the directory:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -b
"ou=People,dc=example,dc=com" -s one -x -E 'sss=cn' -E 'vlv=1/2/70/0' uid
# user069, People, example.com
dn: uid=user069,ou=People,dc=example,dc=com
cn: user069

# user070, People, example.com
dn: uid=user070,ou=People,dc=example,dc=com
cn: user070
```

```
# user071, People, example.com
dn: uid=user071,ou=People,dc=example,dc=com
cn: user071

# user072, People, example.com
dn: uid=user072,ou=People,dc=example,dc=com
cn: user072
```

This example assumes you have entries continuously named **uid=user001** to at least **uid=user072** in **ou=People,dc=example,dc=com**.

Additional resources

- The **-E** parameter description in the **ldapsearch(1)** man page.
- The VLV control in ldapsearch commands

4.4. CREATING A VLV INDEX USING THE WEB CONSOLE TO IMPROVE THE SPEED OF VLV QUERIES

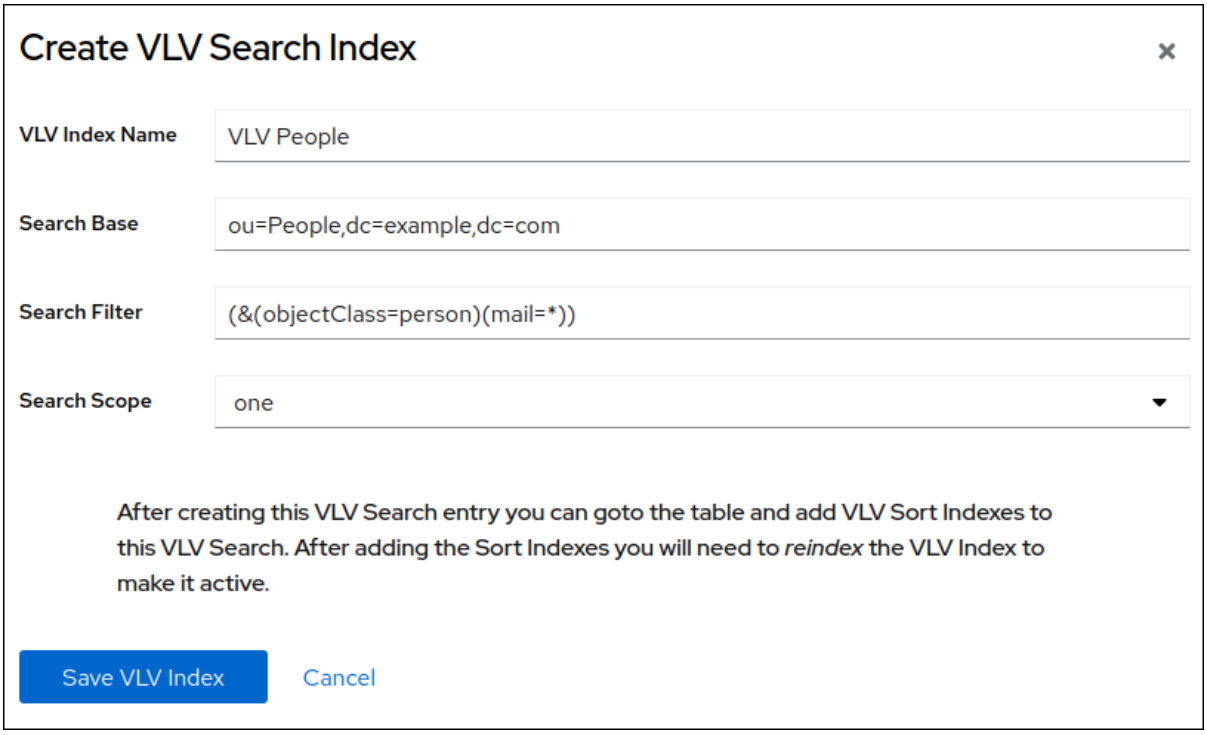
Follow this procedure to create a virtual list view (VLV) index, also called browsing index, for entries in **ou=People,dc=example,dc=com** that contain a **mail** attribute and have the **objectClass** attribute set to **person**.

Prerequisites

- You are logged in to the instance in the web console.
- Your client applications use the VLV control.
- Client applications require to query a contiguous subset of a large search result.
- The directory contains a large number of entries.

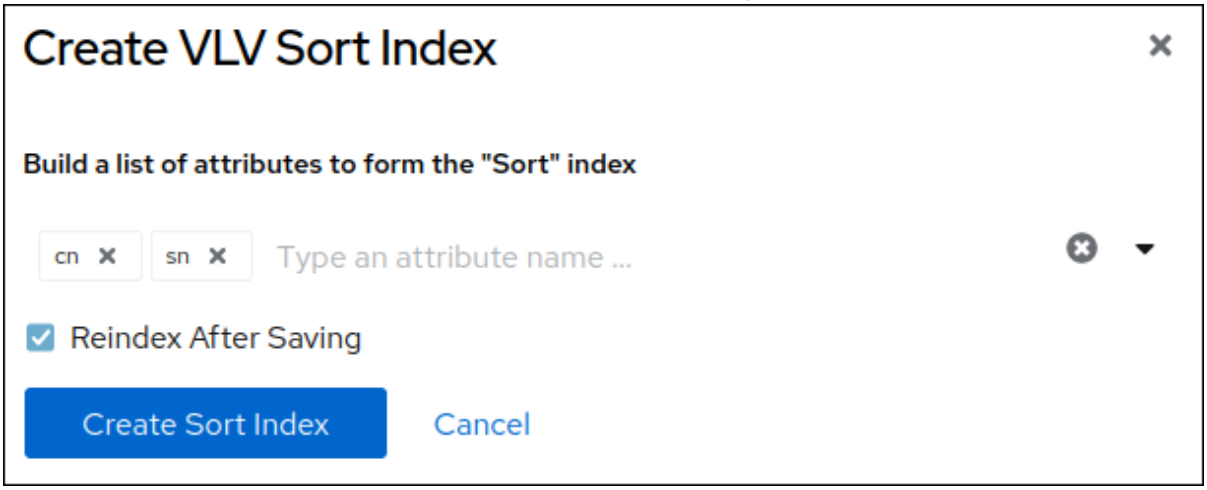
Procedure

1. Navigate to **Database → Suffixes → *dc=example,dc=com* → VLV Indexes**.
2. Click **Create VLV Index**, and fill the fields:



- **VLV Index Name**: The name of the search entry. This can be any name.
- **Search base**: The base DN for the VLV index. Directory Server creates the VLV index on this entry.
- **Search Filter**: The filter Directory Server applies when it creates the VLV index. Only entries that match this filter become part of the index.
- **Search Scope**: The scope of the search to run for entries in the VLV index.

3. Click **Save VLV Index**.
4. Click **Create Sort Index**
5. Enter the attribute names, and select **Reindex After Saving**.



6. Click **Create Sort Index**.

Verification

1. Navigate to **Monitoring → Logging → Errors Log** and verify the successful creation of the VLV index:

```
[26/Nov/2021:11:32:59.001988040 +0100] - INFO - bdb_db2index - userroot: Indexing VLV:
VLV People - cn sn
[26/Nov/2021:11:32:59.507092414 +0100] - INFO - bdb_db2index - userroot: Indexed 1000
entries (2%).
...
[26/Nov/2021:11:33:21.450916820 +0100] - INFO - bdb_db2index - userroot: Indexed 40000
entries (98%).
[26/Nov/2021:11:33:21.671564324 +0100] - INFO - bdb_db2index - userroot: Finished
indexing.
```

2. Use the VLV control in an **ldapsearch** command to query only specific records from the directory:

```
# ldapsearch -D "cn=Directory Manager" -W -H ldap://server.example.com -b
"ou=People,dc=example,dc=com" -s one -x -E 'sss=cn' -E 'vlv=1/2/70/0' uid
# user069, People, example.com
dn: uid=user069,ou=People,dc=example,dc=com
cn: user069

# user070, People, example.com
dn: uid=user070,ou=People,dc=example,dc=com
cn: user070

# user071, People, example.com
dn: uid=user071,ou=People,dc=example,dc=com
cn: user071

# user072, People, example.com
dn: uid=user072,ou=People,dc=example,dc=com
cn: user072
```

This example assumes you have entries continuously named **uid=user001** to at least **uid=user072** in **ou=People,dc=example,dc=com**.

Additional resources

- The **-E** parameter description in the **ldapsearch(1)** man page.
- The VLV control in ldapsearch commands