| Default Value | [none] |
|---|---|

Same as --ndb-connectstring.

- --ndb-nodeid

| Command-Line Format | --ndb-nodeid=# |
|---|---|
| Type | Integer |
| Default Value | [none] |

Set node ID for this node, overriding any ID set by --ndb-connectstring.

- --ndb-optimized-node-selection

| Command-Line Format | --ndb-optimized-node-selection |
|---|---|
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable.

- --no-defaults

| Command-Line Format | --no-defaults |
|---|---|

Do not read default options from any option file other than login file.

- --print-defaults

| Command-Line Format | --print-defaults |
|---|---|

Print program argument list and exit.

- --retries=#, -r

Try to connect this many times before giving up. One connect attempt is made per second.

- --table=*tbl_name*, -t

Specify the table in which to look for an index.

- --unqualified, -u

Use unqualified table names.

- --usage

| Command-Line Format | --usage |
|---|---|

Display help text and exit; same as --help.

- --version

| Command-Line Format | --version |
|---|---|

Display version information and exit.

Table indexes listed in the output are ordered by ID.

## 23.5.10 ndb_drop_index — Drop Index from an NDB Table

`ndb_drop_index` drops the specified index from an `NDB` table. *It is recommended that you use this utility only as an example for writing NDB API applications*—see the Warning later in this section for details.

## Usage

```
ndb_drop_index -c connection_string table_name index -d db_name
```

The statement shown above drops the index named *`index`* from the *`table`* in the *`database`*.

Options that can be used with `ndb_drop_index` are shown in the following table. Additional descriptions follow the table.

**Table 23.32 Command-line options used with the program ndb_drop_index**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database in which table is found | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
|---|---|
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
|---|---|
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
|---|---|
| Type | String |

| Default Value | [none] |
|---|---|

Same as --ndb-connectstring.

- --core-file

| Command-Line Format | --core-file |
|---|---|
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- --database, -d

| Command-Line Format | --database=name |
|---|---|
| Type | String |
| Default Value | TEST_DB |

Name of the database in which the table resides.

- --defaults-extra-file

| Command-Line Format | --defaults-extra-file=path |
|---|---|
| Type | String |
| Default Value | [none] |

Read given file after global files are read.

- --defaults-file

| Command-Line Format | --defaults-file=path |
|---|---|
| Type | String |
| Default Value | [none] |

Read default options from given file only.

- --defaults-group-suffix

| Command-Line Format | --defaults-group-suffix=string |
|---|---|
| Type | String |
| Default Value | [none] |

Also read groups with concat(group, suffix).

- --help

| Command-Line Format | --help |
|---|---|

Display help text and exit.

- --login-path



| Command-Line Format | --login-path=path |
|---|---|
| Type | String |

| Default Value | [none] |
|---|---|

Read given path from login file.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
|---|---|
| Type | String |
| Default Value | [none] |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
|---|---|
| Type | String |
| Default Value | [none] |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
|---|---|
| Type | Integer |
| Default Value | [none] |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
|---|---|
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
|---|---|

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

Print program argument list and exit.

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

**Warning**

*Operations performed on Cluster table indexes using the NDB API are not visible to MySQL and make the table unusable by a MySQL server*. If you use this program to drop an index, then try to access the table from an SQL node, an error results, as shown here:

```
$> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK

NDBT_ProgramExit: 0 - OK

$> ./mysql -u jon -p ctest1
Enter password: *******
Reading table information for completion of table and column names
You can turn off this feature to get a quicker startup with -A

Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 7 to server version: 5.7.42-ndb-7.5.31

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SHOW TABLES;
+------------------+
| Tables_in_ctest1 |
+------------------+
| a                |
| bt1              |
| bt2              |
| dogs             |
| employees        |
| fish             |
+------------------+
6 rows in set (0.00 sec)

mysql> SELECT * FROM dogs;
ERROR 1296 (HY000): Got error 4243 'Index not found' from NDBCLUSTER
```

In such a case, your *only* option for making the table available to MySQL again is to drop the table and re-create it. You can use either the SQL statement`DROP TABLE` or the `ndb_drop_table` utility (see Section 23.5.11, “ndb_drop_table — Drop an NDB Table”) to drop the table.

## 23.5.11 ndb_drop_table — Drop an NDB Table

`ndb_drop_table` drops the specified `NDB` table. (If you try to use this on a table created with a storage engine other than `NDB`, the attempt fails with the error `723: No such table exists`.) This operation is extremely fast; in some cases, it can be an order of magnitude faster than using a MySQL `DROP TABLE` statement on an `NDB` table.

### Usage

```
ndb_drop_table -c connection_string tbl_name -d db_name
```

Options that can be used with `ndb_drop_table` are shown in the following table. Additional descriptions follow the table.

**Table 23.33 Command-line options used with the program ndb_drop_table**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database in which table is found | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--character-sets-dir`

| | |
|---|---|
| Command-Line Format | `--character-sets-dir=path` |
| Removed | 8.0.31 |

  Directory containing character sets.

- `--connect-retries`

| | |
|---|---|
| Command-Line Format | `--connect-retries=#` |
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

  Number of times to retry connection before giving up.

- `--connect-retry-delay`

| | |
|---|---|
| Command-Line Format | `--connect-retry-delay=#` |
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

  Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| | |
|---|---|
| Command-Line Format | `--connect-string=connection_string` |
| Type | String |
| Default Value | `[none]` |

  Same as `--ndb-connectstring`.

- `--core-file`

| | |
|---|---|
| Command-Line Format | `--core-file` |
| Removed | 8.0.31 |

  Write core file on error; used in debugging.

- `--database`, `-d`

| Command-Line Format | `--database=name` |
| --- | --- |
| Type | String |
| Default Value | `TEST_DB` |

Name of the database in which the table resides.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--help`

| Command-Line Format | `--help` |
| --- | --- |

Display help text and exit.

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |



Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
| --- | --- |

Display version information and exit.

## 23.5.12 ndb_error_reporter — NDB Error-Reporting Utility

`ndb_error_reporter` creates an archive from data node and management node log files that can be used to help diagnose bugs or other problems with a cluster. *It is highly recommended that you make use of this utility when filing reports of bugs in NDB Cluster*.

Options that can be used with `ndb_error_reporter` are shown in the following table. Additional descriptions follow the table.

**Table 23.34 Command-line options used with the program ndb_error_reporter**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--connection-timeout=#` | Number of seconds to wait when connecting to nodes before timing out | (Supported in all NDB releases based on MySQL 8.0) |
| `--dry-scp` | Disable scp with remote hosts; used in testing only | (Supported in all NDB releases based on MySQL 8.0) |
| `--fs` | Include file system data in error report; can use a large amount of disk space | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--skip-nodegroup=#` | Skip all nodes in the node group having this ID | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
ndb_error_reporter path/to/config-file [username] [options]
```

This utility is intended for use on a management node host, and requires the path to the management host configuration file (usually named `config.ini`). Optionally, you can supply the name of a user that is able to access the cluster's data nodes using SSH, to copy the data node log files. `ndb_error_reporter` then includes all of these files in archive that is created in the same directory in which it is run. The archive is named `ndb_error_report_YYYYMMDDhhmmss.tar.bz2`, where *`YYYYMMDDhhmmss`* is a datetime string.

`ndb_error_reporter` also accepts the options listed here:

- `--connection-timeout=timeout`

| Command-Line Format | `--connection-timeout=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |

Wait this many seconds when trying to connect to nodes before timing out.

- `--dry-scp`

| Command-Line Format | `--dry-scp` |
|---|---|

Run `ndb_error_reporter` without using scp from remote hosts. Used for testing only.

- `--help`

| Command-Line Format | `--help` |
|---|---|

Display help text and exit.

- `--fs`

| Command-Line Format | `--fs` |
|---|---|

Copy the data node file systems to the management host and include them in the archive.

Because data node file systems can be extremely large, even after being compressed, we ask that you please do *not* send archives created using this option to Oracle unless you are specifically requested to do so.

- `--skip-nodegroup=nodegroup_id`

| Command-Line Format | `--connection-timeout=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |

Skip all nodes belong to the node group having the supplied node group ID.

## 23.5.13 ndb_import — Import CSV Data Into NDB

`ndb_import` imports CSV-formatted data, such as that produced by `mysqldump --tab`, directly into `NDB` using the NDB API. `ndb_import` requires a connection to an NDB management server (`ndb_mgmd`) to function; it does not require a connection to a MySQL Server.

### Usage

```
ndb_import db_name file_name options
```

`ndb_import` requires two arguments. *`db_name`* is the name of the database where the table into which to import the data is found; *`file_name`* is the name of the CSV file from which to read the data; this must include the path to this file if it is not in the current directory. The name of the file must match that of the table; the file's extension, if any, is not taken into consideration. Options supported by `ndb_import` include those for specifying field separators, escapes, and line terminators, and are described later in this section.

Prior to NDB 8.0.30, `ndb_import` rejects any empty lines which it reads from the CSV file. Beginning with NDB 8.0.30, when importing a single column, an empty value that can be used as the column value, ndb_import handles it in the same manner as a `LOAD DATA` statement does.

`ndb_import` must be able to connect to an NDB Cluster management server; for this reason, there must be an unused `[api]` slot in the cluster `config.ini` file.

To duplicate an existing table that uses a different storage engine, such as `InnoDB`, as an `NDB` table, use the `mysql` client to perform a `SELECT INTO OUTFILE` statement to export the existing table to a CSV file, then to execute a `CREATE TABLE LIKE` statement to create a new table having the same structure as the existing table, then perform `ALTER TABLE ... ENGINE=NDB` on the new table; after this, from the system shell, invoke `ndb_import` to load the data into the new `NDB` table. For example, an existing `InnoDB` table named `myinnodb_table` in a database named `myinnodb` can be exported into an `NDB` table named `myndb_table` in a database named `myndb` as shown here, assuming that you are already logged in as a MySQL user with the appropriate privileges:

1. In the `mysql` client:

```
mysql> USE myinnodb;

mysql> SELECT * INTO OUTFILE '/tmp/myndb_table.csv'
     >  FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED BY '"' ESCAPED BY '\\'
     >  LINES TERMINATED BY '\n'
     >  FROM myinnodbtable;

mysql> CREATE DATABASE myndb;

mysql> USE myndb;

mysql> CREATE TABLE myndb_table LIKE myinnodb.myinnodb_table;

mysql> ALTER TABLE myndb_table ENGINE=NDB;
```

```
mysql> EXIT;
Bye
$>
```

Once the target database and table have been created, a running `mysqld` is no longer required. You can stop it using `mysqladmin shutdown` or another method before proceeding, if you wish.

2. In the system shell:

```
# if you are not already in the MySQL bin directory:
$> cd path-to-mysql-bin-dir

$> ndb_import myndb /tmp/myndb_table.csv --fields-optionally-enclosed-by='"' \
    --fields-terminated-by="," --fields-escaped-by='\\'
```

The output should resemble what is shown here:

```
job-1 import myndb.myndb_table from /tmp/myndb_table.csv
job-1 [running] import myndb.myndb_table from /tmp/myndb_table.csv
job-1 [success] import myndb.myndb_table from /tmp/myndb_table.csv
job-1 imported 19984 rows in 0h0m9s at 2277 rows/s
jobs summary: defined: 1 run: 1 with success: 1 with failure: 0
$>
```

All options that can be used with `ndb_import` are shown in the following table. Additional descriptions follow the table.

**Table 23.35 Command-line options used with the program ndb_import**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--abort-on-error` | Dump core on any fatal error; used for debugging | (Supported in all NDB releases based on MySQL 8.0) |
| `--ai-increment=#` | For table with hidden PK, specify autoincrement increment. See mysqld | (Supported in all NDB releases based on MySQL 8.0) |
| `--ai-offset=#` | For table with hidden PK, specify autoincrement offset. See mysqld | (Supported in all NDB releases based on MySQL 8.0) |
| `--ai-prefetch-sz=#` | For table with hidden PK, specify number of autoincrement values that are prefetched. See mysqld | (Supported in all NDB releases based on MySQL 8.0) |
| `--character-sets-dir=path` | Directory containing character sets | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--connections=#` | Number of cluster connections to create | (Supported in all NDB releases based on MySQL 8.0) |
| `--continue` | When job fails, continue to next job | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--csvopt=opts` | Shorthand option for setting typical CSV option values. See documentation for syntax and other information | (Supported in all NDB releases based on MySQL 8.0) |
| `--db-workers=#` | Number of threads, per data node, executing database operations | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--errins-type=name` | Error insert type, for testing purposes; use "list" to obtain all possible values | (Supported in all NDB releases based on MySQL 8.0) |
| `--errins-delay=#` | Error insert delay in milliseconds; random variation is added | (Supported in all NDB releases based on MySQL 8.0) |
| `--fields-enclosed-by=char` | Same as FIELDS ENCLOSED BY option for LOAD DATA statements. For CSV input this is same as using --fields-optionally-enclosed-by | (Supported in all NDB releases based on MySQL 8.0) |
| `--fields-escaped-by=char` | Same as FIELDS ESCAPED BY option for LOAD DATA statements | (Supported in all NDB releases based on MySQL 8.0) |
| `--fields-optionally-enclosed-by=char` | Same as FIELDS OPTIONALLY ENCLOSED BY option for LOAD DATA statements | (Supported in all NDB releases based on MySQL 8.0) |
| `--fields-terminated-by=char` | Same as FIELDS TERMINATED BY option for LOAD DATA statements | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--idlesleep=#` | Number of milliseconds to sleep waiting for more to do | (Supported in all NDB releases based on MySQL 8.0) |
| `--idlespin=#` | Number of times to retry before idlesleep | (Supported in all NDB releases based on MySQL 8.0) |
| `--ignore-lines=#` | Ignore first # lines in input file. Used to skip a non-data header | (Supported in all NDB releases based on MySQL 8.0) |
| `--input-type=name` | Input type: random or csv | (Supported in all NDB releases based on MySQL 8.0) |
| `--input-workers=#` | Number of threads processing input. Must be 2 or more if --input-type is csv | (Supported in all NDB releases based on MySQL 8.0) |
| `--keep-state` | State files (except non-empty *.rej files) are normally removed on job completion. Using this | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| | option causes all state files to be preserved instead | |
| `--lines-terminated-by=char` | Same as LINES TERMINATED BY option for LOAD DATA statements | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--max-rows=#` | Import only this number of input data rows; default is 0, which imports all rows | (Supported in all NDB releases based on MySQL 8.0) |
| `--missing-ai-column='name'` | Indicates that auto-increment values are missing from CSV file to be imported. | ADDED: NDB 8.0.30 |
| `--monitor=#` | Periodically print status of running job if something has changed (status, rejected rows, temporary errors). Value 0 disables. Value 1 prints any change seen. Higher values reduce status printing exponentially up to some pre-defined limit | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-asynch` | Run database operations as batches, in single transactions | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-hint` | Tells transaction coordinator not to use distribution key hint when selecting data node | (Supported in all NDB releases based on MySQL 8.0) |
| `--opbatch=#` | A db execution batch is a set of transactions and operations sent | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| | to NDB kernel. This option limits NDB operations (including blob operations) in a db execution batch. Therefore it also limits number of asynch transactions. Value 0 is not valid | |
| `--opbytes=#` | Limit bytes in execution batch (default 0 = no limit) | (Supported in all NDB releases based on MySQL 8.0) |
| `--output-type=name` | Output type: ndb is default, null used for testing | (Supported in all NDB releases based on MySQL 8.0) |
| `--output-workers=#` | Number of threads processing output or relaying database operations | (Supported in all NDB releases based on MySQL 8.0) |
| `--pagesize=#` | Align I/O buffers to given size | (Supported in all NDB releases based on MySQL 8.0) |
| `--pagecnt=#` | Size of I/O buffers as multiple of page size. CSV input worker allocates double-sized buffer | (Supported in all NDB releases based on MySQL 8.0) |
| `--polltimeout=#` | Timeout per poll for completed asynchonous transactions; polling continues until all polls are completed, or error occurs | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--rejects=#` | Limit number of rejected rows (rows with permanent error) in data load. Default is 0 which means that any rejected row causes a fatal error. The row exceeding the limit is also added to *.rej | (Supported in all NDB releases based on MySQL 8.0) |
| `--resume` | If job aborted (temporary error, user interrupt), resume with rows not yet processed | (Supported in all NDB releases based on MySQL 8.0) |
| `--rowbatch=#` | Limit rows in row queues (default 0 = no limit); must be 1 or more if --input-type is random | (Supported in all NDB releases based on MySQL 8.0) |
| `--rowbytes=#` | Limit bytes in row queues (0 = no limit) | (Supported in all NDB releases based on MySQL 8.0) |
| `--state-dir=path` | Where to write state files; currect directory is default | (Supported in all NDB releases based on MySQL 8.0) |
| `--stats` | Save performance related options and internal statistics in *.sto and *.stt files. These files are kept on successful completion even if --keep-state is not used | (Supported in all NDB releases based on MySQL 8.0) |
| `--table=name`, `-t name` | Name of target to import data into; default is base name of input file | ADDED: NDB 8.0.28 |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--tempdelay=#` | Number of milliseconds to sleep between temporary errors | (Supported in all NDB releases based on MySQL 8.0) |
| `--temperrors=#` | Number of times a transaction can fail due to a temporary error, per execution batch; 0 means any temporary error is fatal. Such errors do not cause any rows to be written to .rej file | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose[=#]`, `-v [#]` | Enable verbose output | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--abort-on-error`

| Command-Line Format | `--abort-on-error` |
|---|---|

Dump core on any fatal error; used for debugging only.

- `--ai-increment=#`

| Command-Line Format | `--ai-increment=#` |
|---|---|
| Type | Integer |
| Default Value | `1` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

For a table with a hidden primary key, specify the autoincrement increment, like the `auto_increment_increment` system variable does in the MySQL Server.

- `--ai-offset=#`

| Command-Line Format | `--ai-offset=#` |
|---|---|
| Type | Integer |
| Default Value | `1` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

For a table with hidden primary key, specify the autoincrement offset. Similar to the `auto_increment_offset` system variable.

- `--ai-prefetch-sz=#`

| Command-Line Format | `--ai-prefetch-sz=#` |
|---|---|
| Type | Integer |
| Default Value | `1024` |

| | |
|---|---|
| Minimum Value | 1 |
| Maximum Value | 4294967295 |

For a table with a hidden primary key, specify the number of autoincrement values that are prefetched. Behaves like the ndb_autoincrement_prefetch_sz system variable does in the MySQL Server.

- --character-sets-dir

| | |
|---|---|
| Command-Line Format | --character-sets-dir=path |

Directory containing character sets.

- --connections=#

| | |
|---|---|
| Command-Line Format | --connections=# |
| Type | Integer |
| Default Value | 1 |
| Minimum Value | 1 |
| Maximum Value | 4294967295 |

Number of cluster connections to create.

- --connect-retries

| | |
|---|---|
| Command-Line Format | --connect-retries=# |
| Type | Integer |
| Default Value | 12 |
| Minimum Value | 0 |
| Maximum Value | 12 |

Number of times to retry connection before giving up.

- --connect-retry-delay

| | |
|---|---|
| Command-Line Format | --connect-retry-delay=# |
| Type | Integer |
| Default Value | 5 |
| Minimum Value | 0 |
| Maximum Value | 5 |

Number of seconds to wait between attempts to contact management server.

- --connect-string

| | |
|---|---|
| Command-Line Format | --connect-string=connection_string |
| Type | String |
| Default Value | [none] |

Same as --ndb-connectstring.

- `--continue`

| Command-Line Format | `--continue` |
|---|---|

When a job fails, continue to the next job.

- `--core-file`

| Command-Line Format | `--core-file` |
|---|---|

Write core file on error; used in debugging.

- `--csvopt=`*`string`*

| Command-Line Format | `--csvopt=opts` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Provides a shortcut method for setting typical CSV import options. The argument to this option is a string consisting of one or more of the following parameters:

- `c`: Fields terminated by comma
- `d`: Use defaults, except where overridden by another parameter
- `n`: Lines terminated by `\n`
- `q`: Fields optionally enclosed by double quote characters (`"`)
- `r`: Line terminated by `\r`

In NDB 8.0.28 and later, the order of parameters used in the argument to this option is handled such that the rightmost parameter always takes precedence over any potentially conflicting parameters which have already been used in the same argument value. This also applies to any duplicate instances of a given parameter. Prior to NDB 8.0.28, the order of the parameters made no difference, other than that, when both `n` and `r` were specified, the one occurring last (rightmost) was the parameter which actually took effect.

This option is intended for use in testing under conditions in which it is difficult to transmit escapes or quotation marks.

- `--db-workers=#`

| Command-Line Format | `--db-workers=#` |
|---|---|
| Type | Integer |
| Default Value | `4` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

Number of threads, per data node, executing database operations.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |

| Default Value | [none] |
|---|---|

Read default options from given file only.

- `--defaults-extra-file`

| Command-Line Format | --defaults-extra-file=path |
|---|---|
| Type | String |
| Default Value | [none] |

Read given file after global files are read.

- `--defaults-group-suffix`

| Command-Line Format | --defaults-group-suffix=string |
|---|---|
| Type | String |
| Default Value | [none] |

Also read groups with concat(group, suffix).

- `--errins-type=`*`name`*

| Command-Line Format | --errins-type=name |
|---|---|
| Type | Enumeration |
| Default Value | [none] |
| Valid Values | stopjob<br>stopall<br>sighup<br>sigint<br>list |

Error insert type; use `list` as the *`name`* value to obtain all possible values. This option is used for testing purposes only.

- `--errins-delay=`*`#`*

| Command-Line Format | --errins-delay=# |
|---|---|
| Type | Integer |
| Default Value | 1000 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |
| Unit | ms |

Error insert delay in milliseconds; random variation is added. This option is used for testing purposes only.

- `--fields-enclosed-by=`*`char`*



| Command-Line Format | --fields-enclosed-by=char |
|---|---|
| Type | String |

| Default Value | [none] |
|---|---|

This works in the same way as the FIELDS ENCLOSED BY option does for the LOAD DATA statement, specifying a character to be interpreted as quoting field values. For CSV input, this is the same as --fields-optionally-enclosed-by.

- --fields-escaped-by=*name*

| Command-Line Format | --fields-escaped-by=char |
|---|---|
| Type | String |
| Default Value | \ |

Specify an escape character in the same way as the FIELDS ESCAPED BY option does for the SQL LOAD DATA statement.

- --fields-optionally-enclosed-by=*char*

| Command-Line Format | --fields-optionally-enclosed-by=char |
|---|---|
| Type | String |
| Default Value | [none] |

This works in the same way as the FIELDS OPTIONALLY ENCLOSED BY option does for the LOAD DATA statement, specifying a character to be interpreted as optionally quoting field values. For CSV input, this is the same as --fields-enclosed-by.

- --fields-terminated-by=*char*

| Command-Line Format | --fields-terminated-by=char |
|---|---|
| Type | String |
| Default Value | \t |

This works in the same way as the FIELDS TERMINATED BY option does for the LOAD DATA statement, specifying a character to be interpreted as the field separator.

- --help

| Command-Line Format | --help |
|---|---|

Display help text and exit.

- --idlesleep=#

| Command-Line Format | --idlesleep=# |
|---|---|
| Type | Integer |
| Default Value | 1 |
| Minimum Value | 1 |
| Maximum Value | 4294967295 |
| Unit | ms |

Number of milliseconds to sleep waiting for more work to perform.

- --idlespin=#

| Command-Line Format | --idlespin=# |
|---|---|
| Type | Integer |

| Default Value | 0 |
| --- | --- |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |

Number of times to retry before sleeping.

- `--ignore-lines=#`

| Command-Line Format | --ignore-lines=# |
| --- | --- |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |

Cause ndb_import to ignore the first `#` lines of the input file. This can be employed to skip a file header that does not contain any data.

- `--input-type=name`

| Command-Line Format | --input-type=name |
| --- | --- |
| Type | Enumeration |
| Default Value | csv |
| Valid Values | random<br><br>csv |

Set the type of input type. The default is `csv`; `random` is intended for testing purposes only. .

- `--input-workers=#`

| Command-Line Format | --input-workers=# |
| --- | --- |
| Type | Integer |
| Default Value | 4 |
| Minimum Value | 1 |
| Maximum Value | 4294967295 |

Set the number of threads processing input.

- `--keep-state`

| Command-Line Format | --keep-state |
| --- | --- |

By default, ndb_import removes all state files (except non-empty `*.rej` files) when it completes a job. Specify this option (nor argument is required) to force the program to retain all state files instead.

- `--lines-terminated-by=name`

| Command-Line Format | --lines-terminated-by=char |
| --- | --- |
| Type | String |
| Default Value | \n |

This works in the same way as the `LINES TERMINATED BY` option does for the `LOAD DATA` statement, specifying a character to be interpreted as end-of-line.

- `--log-level=#`

| Command-Line Format | `--log-level=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `2` |

Performs internal logging at the given level. This option is intended primarily for internal and development use.

In debug builds of NDB only, the logging level can be set using this option to a maximum of 4.

- `--login-path`

| Command-Line Format | `--login-path=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--max-rows=#`

| Command-Line Format | `--max-rows=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | bytes |

Import only this number of input data rows; the default is 0, which imports all rows.

- `--missing-ai-column`

| Command-Line Format | `--missing-ai-column='name'` |
|---|---|
| Introduced | 8.0.30-ndb-8.0.30 |
| Type | Boolean |
| Default Value | `FALSE` |

This option can be employed when importing a single table, or multiple tables. When used, it indicates that the CSV file being imported does not contain any values for an `AUTO_INCREMENT` column, and that `ndb_import` should supply them; if the option is used and the `AUTO_INCREMENT` column contains any values, the import operation cannot proceed.

- `--monitor=#`

| Command-Line Format | `--monitor=#` |
|---|---|
| Type | Integer |
| Default Value | `2` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | bytes |

Periodically print the status of a running job if something has changed (status, rejected rows, temporary errors). Set to 0 to disable this reporting. Setting to 1 prints any change that is seen. Higher values reduce the frequency of this status reporting.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-asynch`

| Command-Line Format | `--no-asynch` |
| --- | --- |

Run database operations as batches, in single transactions.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--no-hint`

| Command-Line Format | `--no-hint` |
| --- | --- |

Do not use distribution key hinting to select a data node.

- `--opbatch=#`

| Command-Line Format | `--opbatch=#` |
| --- | --- |
| Type | Integer |
| Default Value | `256` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |
| Unit | bytes |

Set a limit on the number of operations (including blob operations), and thus the number of asynchronous transactions, per execution batch.

- `--opbytes=#`

| Command-Line Format | `--opbytes=#` |
| --- | --- |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | bytes |

Set a limit on the number of bytes per execution batch. Use 0 for no limit.

- `--output-type=name`

| Command-Line Format | `--output-type=name` |
| --- | --- |
| Type | Enumeration |
| Default Value | `ndb` |
| Valid Values | `null` |

Set the output type. `ndb` is the default. `null` is used only for testing.

- `--output-workers=#`

| Command-Line Format | `--output-workers=#` |
| --- | --- |
| Type | Integer |
| Default Value | `2` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

Set the number of threads processing output or relaying database operations.

- `--pagesize=#`

| Command-Line Format | `--pagesize=#` |
| --- | --- |
| Type | Integer |
| Default Value | `4096` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

| Unit | bytes |
|---|---|

Align I/O buffers to the given size.

- `--pagecnt=#`

| Command-Line Format | `--pagecnt=#` |
|---|---|
| Type | Integer |
| Default Value | `64` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |

Set the size of I/O buffers as multiple of page size. The CSV input worker allocates buffer that is doubled in size.

- `--polltimeout=#`

| Command-Line Format | `--polltimeout=#` |
|---|---|
| Type | Integer |
| Default Value | `1000` |
| Minimum Value | `1` |
| Maximum Value | `4294967295` |
| Unit | ms |

Set a timeout per poll for completed asynchronous transactions; polling continues until all polls are completed, or until an error occurs.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

Print program argument list and exit.

- `--rejects=#`

| Command-Line Format | `--rejects=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |

Limit the number of rejected rows (rows with permanent errors) in the data load. The default is 0, which means that any rejected row causes a fatal error. Any rows causing the limit to be exceeded are added to the `.rej` file.

The limit imposed by this option is effective for the duration of the current run. A run restarted using `--resume` is considered a “new” run for this purpose.

- `--resume`

| Command-Line Format | `--resume` |
|---|---|

If a job is aborted (due to a temporary db error or when interrupted by the user), resume with any rows not yet processed.

- `--rowbatch=#`

| Command-Line Format | `--rowbatch=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | rows |

Set a limit on the number of rows per row queue. Use 0 for no limit.

- `--rowbytes=#`

| Command-Line Format | `--rowbytes=#` |
|---|---|
| Type | Integer |
| Default Value | `262144` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | bytes |

Set a limit on the number of bytes per row queue. Use 0 for no limit.

- `--stats`

| Command-Line Format | `--stats` |
|---|---|

Save information about options related to performance and other internal statistics in files named `*.sto` and `*.stt`. These files are always kept on successful completion (even if `--keep-state` is not also specified).

- `--state-dir=name`

| Command-Line Format | `--state-dir=path` |
|---|---|
| Type | String |
| Default Value | `.` |

Where to write the state files (*`tbl_name`*`.map`, *`tbl_name`*`.rej`, *`tbl_name`*`.res`, and *`tbl_name`*`.stt`) produced by a run of the program; the default is the current directory.

- `--table=name`

| Command-Line Format | `--table=name` |
|---|---|
| Introduced | 8.0.28-ndb-8.0.28 |
| Type | String |
| Default Value | `[input file base name]` |

By default, `ndb_import` attempts to import data into a table whose name is the base name of the CSV file from which the data is being read. Beginning with NDB 8.0.28, you can override the choice of table name by specifying it using the `--table` option (short form `-t`).

- `--tempdelay=#`

| Command-Line Format | `--tempdelay=#` |
|---|---|

| Type | Integer |
|---|---|
| Default Value | `10` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |
| Unit | ms |

Number of milliseconds to sleep between temporary errors.

- `--temperrors=#`

| Command-Line Format | `--temperrors=#` |
|---|---|
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `4294967295` |

Number of times a transaction can fail due to a temporary error, per execution batch. The default is 0, which means that any temporary error is fatal. Temporary errors do not cause any rows to be added to the `.rej` file.

- `--verbose`, `-v`

| Command-Line Format | `--verbose[=#]` |
|---|---|
| Type | Boolean |
| Default Value | `false` |

Enable verbose output.

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

As with `LOAD DATA`, options for field and line formatting much match those used to create the CSV file, whether this was done using `SELECT INTO ... OUTFILE`, or by some other means. There is no equivalent to the `LOAD DATA` statement `STARTING WITH` option.

## 23.5.14 ndb_index_stat — NDB Index Statistics Utility

`ndb_index_stat` provides per-fragment statistical information about indexes on `NDB` tables. This includes cache version and age, number of index entries per partition, and memory consumption by indexes.

### Usage

To obtain basic index statistics about a given `NDB` table, invoke `ndb_index_stat` as shown here, with the name of the table as the first argument and the name of the database containing this table specified immediately following it, using the `--database` (`-d`) option:

```
ndb_index_stat table -d database
```

In this example, we use `ndb_index_stat` to obtain such information about an `NDB` table named `mytable` in the `test` database:

```
$> ndb_index_stat -d test mytable
table:City index:PRIMARY fragCount:2
sampleVersion:3 loadTime:1399585986 sampleCount:1994 keyBytes:7976
query cache: valid:1 sampleCount:1994 totalBytes:27916
times in ms: save: 7.133 sort: 1.974 sort per sample: 0.000

NDBT_ProgramExit: 0 - OK
```

`sampleVersion` is the version number of the cache from which the statistics data is taken. Running `ndb_index_stat` with the `--update` option causes sampleVersion to be incremented.

`loadTime` shows when the cache was last updated. This is expressed as seconds since the Unix Epoch.

`sampleCount` is the number of index entries found per partition. You can estimate the total number of entries by multiplying this by the number of fragments (shown as `fragCount`).

`sampleCount` can be compared with the cardinality of `SHOW INDEX` or `INFORMATION_SCHEMA.STATISTICS`, although the latter two provide a view of the table as a whole, while `ndb_index_stat` provides a per-fragment average.

`keyBytes` is the number of bytes used by the index. In this example, the primary key is an integer, which requires four bytes for each index, so `keyBytes` can be calculated in this case as shown here:

```
    keyBytes = sampleCount * (4 bytes per index) = 1994 * 4 = 7976
```

This information can also be obtained using the corresponding column definitions from `INFORMATION_SCHEMA.COLUMNS` (this requires a MySQL Server and a MySQL client application).

`totalBytes` is the total memory consumed by all indexes on the table, in bytes.

Timings shown in the preceding examples are specific to each invocation of `ndb_index_stat`.

The `--verbose` option provides some additional output, as shown here:

```
$> ndb_index_stat -d test mytable --verbose
random seed 1337010518
connected
loop 1 of 1
table:mytable index:PRIMARY fragCount:4
sampleVersion:2 loadTime:1336751773 sampleCount:0 keyBytes:0
read stats
query cache created
query cache: valid:1 sampleCount:0 totalBytes:0
times in ms: save: 20.766 sort: 0.001
disconnected

NDBT_ProgramExit: 0 - OK

$>
```

If the only output from the program is `NDBT_ProgramExit: 0 - OK`, this may indicate that no statistics yet exist. To force them to be created (or updated if they already exist), invoke `ndb_index_stat` with the `--update` option, or execute `ANALYZE TABLE` on the table in the `mysql` client.

## Options

The following table includes options that are specific to the NDB Cluster `ndb_index_stat` utility. Additional descriptions are listed following the table.

**Table 23.36 Command-line options used with the program ndb_index_stat**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database containing table | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--delete` | Delete index statistics for table, stopping any auto-update previously configured | (Supported in all NDB releases based on MySQL 8.0) |
| `--dump` | Print query cache | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--loops=#` | Set the number of times to perform given command; default is 0 | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--query=#` | Perform random range queries on first key attr (must be int unsigned) | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-drop` | Drop any statistics tables and events in NDB kernel (all statistics are lost) | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-create` | Create all statistics tables and events in NDB kernel, if none of them already exist | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-create-if-not-exist` | Create any statistics tables and events in NDB kernel that do not already exist | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-create-if-not-valid` | Create any statistics tables or events that do not already exist in the NDB kernel, after dropping any that are invalid | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-check` | Verify that NDB system index statistics and event tables exist | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-skip-tables` | Do not apply sys-* options to tables | (Supported in all NDB releases based on MySQL 8.0) |
| `--sys-skip-events` | Do not apply sys-* options to events | (Supported in all NDB releases based on MySQL 8.0) |
| `--update` | Update index statistics for table, restarting any auto-update previously configured | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose`, `-v` | Turn on verbose output | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
| --- | --- |
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
| --- | --- |
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
| --- | --- |
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--database=`*`name`*, `-d` *`name`*

| Command-Line Format | `--database=name` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |
| Minimum Value | |
| Maximum Value | |

The name of the database that contains the table being queried.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--delete`

| Command-Line Format | `--delete` |
|---|---|

Delete the index statistics for the given table, stopping any auto-update that was previously configured.

- `--dump`

| Command-Line Format | `--dump` |
|---|---|

Dump the contents of the query cache.

- `--help`

| Command-Line Format | `--help` |
|---|---|

Display help text and exit.

- `--login-path`

| Command-Line Format | `--login-path=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--loops=#`

| Command-Line Format | `--loops=#` |
|---|---|
| Type | Numeric |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `MAX_INT` |

Repeat commands this number of times (for use in testing).

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--query=#`

| Command-Line Format | `--query=#` |
| --- | --- |
| Type | Numeric |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `MAX_INT` |

Perform random range queries on first key attribute (must be int unsigned).

- `--sys-drop`

| Command-Line Format | `--sys-drop` |
|---|---|

Drop all statistics tables and events in the NDB kernel. *This causes all statistics to be lost*.

- `--sys-create`

| Command-Line Format | `--sys-create` |
|---|---|

Create all statistics tables and events in the NDB kernel. This works only if none of them exist previously.

- `--sys-create-if-not-exist`

| Command-Line Format | `--sys-create-if-not-exist` |
|---|---|

Create any NDB system statistics tables or events (or both) that do not already exist when the program is invoked.

- `--sys-create-if-not-valid`

| Command-Line Format | `--sys-create-if-not-valid` |
|---|---|

Create any NDB system statistics tables or events that do not already exist, after dropping any that are invalid.

- `--sys-check`

| Command-Line Format | `--sys-check` |
|---|---|

Verify that all required system statistics tables and events exist in the NDB kernel.

- `--sys-skip-tables`

| Command-Line Format | `--sys-skip-tables` |
|---|---|

Do not apply any `--sys-*` options to any statistics tables.

- `--sys-skip-events`

| Command-Line Format | `--sys-skip-events` |
|---|---|

Do not apply any `--sys-*` options to any events.

- `--update`

| Command-Line Format | `--update` |
|---|---|

Update the index statistics for the given table, and restart any auto-update that was previously configured.

- `--usage`



| Command-Line Format | `--usage` |
|---|---|

- `--verbose`

| Command-Line Format | `--verbose` |
|---|---|

  Turn on verbose output.

- `--version`

| Command-Line Format | `--version` |
|---|---|

  Display version information and exit.

**ndb_index_stat system options.** The following options are used to generate and update the statistics tables in the NDB kernel. None of these options can be mixed with statistics options (see ndb_index_stat statistics options).

- `--sys-drop`
- `--sys-create`
- `--sys-create-if-not-exist`
- `--sys-create-if-not-valid`
- `--sys-check`
- `--sys-skip-tables`
- `--sys-skip-events`

**ndb_index_stat statistics options.** The options listed here are used to generate index statistics. They work with a given table and database. They cannot be mixed with system options (see ndb_index_stat system options).

- `--database`
- `--delete`
- `--update`
- `--dump`
- `--query`

## 23.5.15 ndb_move_data — NDB Data Copy Utility

`ndb_move_data` copies data from one NDB table to another.

### Usage

The program is invoked with the names of the source and target tables; either or both of these may be qualified optionally with the database name. Both tables must use the NDB storage engine.

```
ndb_move_data options source target
```

Options that can be used with `ndb_move_data` are shown in the following table. Additional descriptions follow the table.

**Table 23.37 Command-line options used with the program ndb_move_data**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--abort-on-error` | Dump core on permanent error (debug option) | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--character-sets-dir=path` | Directory where character sets are | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database in which table is found | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--drop-source` | Drop source table after all rows have been moved | (Supported in all NDB releases based on MySQL 8.0) |
| `--error-insert` | Insert random temporary errors (used in testing) | (Supported in all NDB releases based on MySQL 8.0) |
| `--exclude-missing-columns` | Ignore extra columns in source or target table | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--lossy-conversions`, `-l` | Allow attribute data to be truncated when converted to smaller type | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--promote-attributes`, `-A` | Allow attribute data to be converted to larger type | (Supported in all NDB releases based on MySQL 8.0) |
| `--staging-tries=x[,y[,z]]` | Specify tries on temporary errors; format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms) | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose` | Enable verbose messages | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--abort-on-error`

| Command-Line Format | `--abort-on-error` |
|---|---|

  Dump core on permanent error (debug option).

- `--character-sets-dir=`*`name`*

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |
| Type | String |
| Default Value | `[none]` |

  Directory where character sets are.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
|---|---|
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

  Number of seconds to wait between attempts to contact management server.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
|---|---|
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
|---|---|
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--database`=*`dbname`*, `-d`

| Command-Line Format | `--database=name` |
|---|---|
| Type | String |
| Default Value | `TEST_DB` |

Name of the database in which the table is found.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
|---|---|
| Type | String |

| Default Value | [none] |
|---|---|

Also read groups with concat(group, suffix).

- `--drop-source`

| Command-Line Format | --drop-source |
|---|---|

Drop source table after all rows have been moved.

- `--error-insert`

| Command-Line Format | --error-insert |
|---|---|

Insert random temporary errors (testing option).

- `--exclude-missing-columns`

| Command-Line Format | --exclude-missing-columns |
|---|---|

Ignore extra columns in source or target table.

- `--help`

| Command-Line Format | --help |
|---|---|

Display help text and exit.

- `--login-path`

| Command-Line Format | --login-path=path |
|---|---|
| Type | String |
| Default Value | [none] |

Read given path from login file.

- `--lossy-conversions`, `-l`

| Command-Line Format | --lossy-conversions |
|---|---|

Allow attribute data to be truncated when converted to a smaller type.

- `--ndb-connectstring`

| Command-Line Format | --ndb-connectstring=connection_string |
|---|---|
| Type | String |
| Default Value | [none] |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | --ndb-mgmd-host=connection_string |
|---|---|
| Type | String |
| Default Value | [none] |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--promote-attributes`, `-A`

| Command-Line Format | `--promote-attributes` |
| --- | --- |

Allow attribute data to be converted to a larger type.

- `--staging-tries`=*`x[,y[,z]]`*

| Command-Line Format | `--staging-tries=x[,y[,z]]` |
| --- | --- |
| Type | String |
| Default Value | `0,1000,60000` |

Specify tries on temporary errors. Format is x[,y[,z]] where x=max tries (0=no limit), y=min delay (ms), z=max delay (ms).

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as `--help`.

- `--verbose`

| Command-Line Format | `--verbose` |
| --- | --- |

Enable verbose messages.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

## 23.5.16 ndb_perror — Obtain NDB Error Message Information

`ndb_perror` shows information about an NDB error, given its error code. This includes the error message, the type of error, and whether the error is permanent or temporary. This is intended as a drop-in replacement for `perror --ndb`, which is no longer supported.

### Usage

```
ndb_perror [options] error_code
```

`ndb_perror` does not need to access a running NDB Cluster, or any nodes (including SQL nodes). To view information about a given NDB error, invoke the program, using the error code as an argument, like this:

```
$> ndb_perror 323
NDB error code 323: Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

To display only the error message, invoke `ndb_perror` with the `--silent` option (short form `-s`), as shown here:

```
$> ndb_perror -s 323
Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

Like `perror`, `ndb_perror` accepts multiple error codes:

```
$> ndb_perror 321 1001
NDB error code 321: Invalid nodegroup id: Permanent error: Application error
NDB error code 1001: Illegal connect string
```

Additional program options for `ndb_perror` are described later in this section.

`ndb_perror` replaces `perror --ndb`, which is no longer supported by NDB Cluster. To make substitution easier in scripts and other applications that might depend on `perror` for obtaining NDB error information, `ndb_perror` supports its own “dummy” `--ndb` option, which does nothing.

The following table includes all options that are specific to the NDB Cluster program `ndb_perror`. Additional descriptions follow the table.

**Table 23.38 Command-line options used with the program ndb_perror**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb` | For compatibility with applications depending on old versions of perror; does nothing | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--silent`, `-s` | Show error message only | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Print program version information and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose`, `-v` | Verbose output; disable with --silent | (Supported in all NDB releases based on MySQL 8.0) |

## Additional Options

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--help`, `-?`

| Command-Line Format | `--help` |
|---|---|

Display program help text and exit.

- `--login-path`

| Command-Line Format | `--login-path=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--ndb`

| Command-Line Format | `--ndb` |
|---|---|

For compatibility with applications depending on old versions of `perror` that use that program's `--ndb` option. The option when used with `ndb_perror` does nothing, and is ignored by it.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
|---|---|

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

Print program argument list and exit.

- `--silent`, `-s`

| Command-Line Format | `--silent` |
|---|---|

Show error message only.

- `--version`, `-V`

| Command-Line Format | `--version` |
|---|---|

Print program version information and exit.

- `--verbose`, `-v`

| Command-Line Format | `--verbose` |
|---|---|

Verbose output; disable with `--silent`.

## 23.5.17 ndb_print_backup_file — Print NDB Backup File Contents

`ndb_print_backup_file` obtains diagnostic information from a cluster backup file.

**Table 23.39 Command-line options used with the program ndb_print_backup_file**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--backup-key=key`, `-K password` | Use this password to decrypt file | ADDED: NDB 8.0.31 |
| `--backup-key-from-stdin` | Get decryption key in a secure fashion from STDIN | ADDED: NDB 8.0.31 |
| `--backup-password=password`, `-P password` | Use this password to decrypt file | ADDED: NDB 8.0.22 |
| `--backup-password-from-stdin` | Get decryption password in a secure fashion from STDIN | ADDED: NDB 8.0.24 |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--control-directory-number=#`,<br>`-c #` | Control directory number | ADDED: NDB 8.0.24 |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--fragment-id=#`,<br>`-f #` | Fragment ID | ADDED: NDB 8.0.24 |
| `--help`,<br>`--usage`,<br>`-h`,<br>`-?` | Print usage information | ADDED: NDB 8.0.24 |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-print-rows`,<br>`-u` | Do not print rows | ADDED: NDB 8.0.24 |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-header-words`,<br>`-h` | Print header words | ADDED: NDB 8.0.24 |
| `--print-restored-rows` | Print restored rows | ADDED: NDB 8.0.24 |
| `--print-rows`,<br>`-U` | Print rows. Enabled by default; disable with --no-print-rows | ADDED: NDB 8.0.24 |
| `--print-rows-per-page` | Print rows per page | ADDED: NDB 8.0.24 |
| `--rowid-file=path`,<br>`-n path` | File containing row ID to check for | ADDED: NDB 8.0.24 |
| `--show-ignored-rows`,<br>`-i` | Show ignored rows | ADDED: NDB 8.0.24 |
| `--table-id=#`,<br>`-t #` | Table ID; used with --print-restored rows | ADDED: NDB 8.0.24 |
| `--usage`,<br>`-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose[=#]`, | Verbosity level | ADDED: NDB 8.0.24 |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-v`<br>`--version`,<br>`-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
ndb_print_backup_file [-P password] file_name
```

*`file_name`* is the name of a cluster backup file. This can be any of the files (`.Data`, `.ctl`, or `.log` file) found in a cluster backup directory. These files are found in the data node's backup directory under the subdirectory `BACKUP-#`, where `#` is the sequence number for the backup. For more information about cluster backup files and their contents, see Section 23.6.8.1, “NDB Cluster Backup Concepts”.

Like `ndb_print_schema_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

In NDB 8.0, this program can also be used to read undo log files.

## Options

Prior to NDB 8.0.24, `ndb_print_backup_file` supported only the `-P` option. Beginning with NDB 8.0.24, the program supports a number of options, which are described in the following list.

- `--backup-key`, `-K`

| Command-Line Format | `--backup-key=key` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Specify the key needed to decrypt an encrypted backup.

- `--backup-key-from-stdin`

| Command-Line Format | `--backup-key-from-stdin` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Allow input of the decryption key from standard input, similar to entering a password after invoking `mysql --password` with no password supplied.

- `--backup-password`

| Command-Line Format | `--backup-password=password` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |
| Type | String |
| Default Value | `[none]` |

Specify the password needed to decrypt an encrypted backup.

The long form of this option is available beginning with NDB 8.0.24.

- `--backup-password-from-stdin`

| Command-Line Format | `--backup-password-from-stdin` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |

Allow input of the password from standard input, similar to entering a password after invoking `mysql --password` with no password supplied.

- `--control-directory-number`

| Command-Line Format | `--control-directory-number=#` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |
| Type | Integer |
| Default Value | `0` |

Control file directory number. Used together with `--print-restored-rows`.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--fragment-id`

| Command-Line Format | `--fragment-id=#` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |
| Type | Integer |
| Default Value | `0` |

Fragment ID. Used together with `--print-restored-rows`.

- `--help`

| Command-Line Format | `--help`<br><br>`--usage` |
| --- | --- |

| Introduced | 8.0.24-ndb-8.0.24 |
|---|---|

Print program usage information.

- `--login-path`

| Command-Line Format | `--login-path=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
|---|---|

Do not read default options from any option file other than login file.

- `--no-print-rows`

| Command-Line Format | `--no-print-rows` |
|---|---|
| Introduced | 8.0.24-ndb-8.0.24 |

Do not include rows in output.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

Print program argument list and exit.

- `--print-header-words`

| Command-Line Format | `--print-header-words` |
|---|---|
| Introduced | 8.0.24-ndb-8.0.24 |

Include header words in output.

- `--print-restored-rows`

| Command-Line Format | `--print-restored-rows` |
|---|---|
| Introduced | 8.0.24-ndb-8.0.24 |

Include restored rows in output, using the file `LCP/`*`c`*`/T`*`t`*`F`*`f`*`.ctl`, for which the values are set as follows:

- *`c`* is the control file number set using `--control-directory-number`
- *`t`* is the table ID set using `--table-id`
- *`f`* is the fragment ID set using `--fragment-id`

- `--print-rows`

| Command-Line Format | `--print-rows` |
|---|---|
| Introduced | 8.0.24-ndb-8.0.24 |

Print rows. This option is enabled by default; to disable it, use `--no-print-rows`.

- `--print-rows-per-page`

| Command-Line Format | `--print-rows-per-page` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |

Print rows per page.

- `--rowid-file`

| Command-Line Format | `--rowid-file=path` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |
| Type | File name |
| Default Value | `[none]` |

File to check for row ID.

- `--show-ignored-rows`

| Command-Line Format | `--show-ignored-rows` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |

Show ignored rows.

- `--table-id`

| Command-Line Format | `--table-id=#` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |
| Type | Integer |
| Default Value | `[none]` |

Table ID. Used together with `--print-restored-rows`.

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as `--help`.

- `--verbose`

| Command-Line Format | `--verbose[=#]` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |
| Type | Integer |
| Default Value | `0` |

Verbosity level of output. A greater value indicates increased verbosity.

- `--version`

| Command-Line Format | `--version` |
| --- | --- |

Display version information and exit.

## 23.5.18 ndb_print_file — Print NDB Disk Data File Contents

`ndb_print_file` obtains information from an NDB Cluster Disk Data file.

## Usage

```
ndb_print_file [-v] [-q] file_name+
```

*`file_name`* is the name of an NDB Cluster Disk Data file. Multiple filenames are accepted, separated by spaces.

Like `ndb_print_schema_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_file` must be run on an NDB Cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

## Options

**Table 23.40 Command-line options used with the program ndb_print_file**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--file-key=hex_data`, `-K hex_data` | Supply encryption key using stdin, tty, or my.cnf file | ADDED: NDB 8.0.31 |
| `--file-key-from-stdin` | Supply encryption key using stdin | ADDED: NDB 8.0.31 |
| `--help`, `-?` | Display help text and exit; same as --usage | (Supported in all NDB releases based on MySQL 8.0) |
| `--quiet`, `-q` | Reduce verbosity of output | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose`, `-v` | Increase verbosity of output | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

`ndb_print_file` supports the following options:

- `--file-key`, `-K`

| Command-Line Format | `--file-key=hex_data` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

  Supply file system encryption or decryption key from `stdin`, `tty`, or a `my.cnf` file.

- `--file-key-from-stdin`

| Command-Line Format | `--file-key-from-stdin` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |
| Type | Boolean |
| Default Value | `FALSE` |
| Valid Values | `TRUE` |

Supply file system encryption or decryption key from `stdin`.

- `--help`, `-h`, `-?`

| Command-Line Format | `--help` |
|---|---|

Print help message and exit.

- `--quiet`, `-q`

| Command-Line Format | `--quiet` |
|---|---|

Suppress output (quiet mode).

- `--usage`, `-?`

| Command-Line Format | `--usage` |
|---|---|

Print help message and exit.

- `--verbose`, `-v`

| Command-Line Format | `--verbose` |
|---|---|

Make output verbose.

- `--version`, `-v`

| Command-Line Format | `--version` |
|---|---|

Print version information and exit.

For more information, see Section 23.6.11, “NDB Cluster Disk Data Tables”.

## 23.5.19 ndb_print_frag_file — Print NDB Fragment List File Contents

`ndb_print_frag_file` obtains information from a cluster fragment list file. It is intended for use in helping to diagnose issues with data node restarts.

### Usage

```
ndb_print_frag_file file_name
```

*`file_name`* is the name of a cluster fragment list file, which matches the pattern `S`*`X`*`.FragList`, where *`X`* is a digit in the range 2-9 inclusive, and are found in the data node file system of the data node having the node ID *`nodeid`*, in directories named `ndb_`*`nodeid`*`_fs/DN/DBDIH/`, where *`N`* is `1` or `2`. Each fragment file contains records of the fragments belonging to each `NDB` table. For more information about cluster fragment files, see NDB Cluster Data Node File System Directory.

Like `ndb_print_backup_file`, `ndb_print_sys_file`, and `ndb_print_schema_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server), `ndb_print_frag_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

### Additional Options

None.

### Sample Output

```
$> ndb_print_frag_file /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList
Filename: /usr/local/mysqld/data/ndb_3_fs/D1/DBDIH/S2.FragList with size 8192
noOfPages = 1 noOfWords = 182
Table Data
----------
Num Frags: 2 NoOfReplicas: 2 hashpointer: 4294967040
kvalue: 6 mask: 0x00000000 method: HashMap
Storage is on Logged and checkpointed, survives SR
------ Fragment with FragId: 0 --------
Preferred Primary: 2 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
------ Fragment with FragId: 1 --------
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 1
-------Stored Replica----------
Replica node is: 3 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
-------Stored Replica----------
Replica node is: 2 initialGci: 2 numCrashedReplicas = 0 nextLcpNo = 1
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
```

## 23.5.20 ndb_print_schema_file — Print NDB Schema File Contents

`ndb_print_schema_file` obtains diagnostic information from a cluster schema file.

### Usage

```
ndb_print_schema_file file_name
```

*`file_name`* is the name of a cluster schema file. For more information about cluster schema files, see NDB Cluster Data Node File System Directory.

Like `ndb_print_backup_file` and `ndb_print_sys_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_schema_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

### Additional Options

None.

## 23.5.21 ndb_print_sys_file — Print NDB System File Contents

`ndb_print_sys_file` obtains diagnostic information from an NDB Cluster system file.

### Usage

```
ndb_print_sys_file file_name
```

*`file_name`* is the name of a cluster system file (sysfile). Cluster system files are located in a data node's data directory (`DataDir`); the path under this directory to system files matches the pattern `ndb_#_fs/D#/DBDIH/P#.sysfile`. In each case, the `#` represents a number (not necessarily the same number). For more information, see NDB Cluster Data Node File System Directory.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the other `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_print_backup_file` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

### Additional Options

None.

## 23.5.22 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log

Reads a redo log file, checking it for errors, printing its contents in a human-readable format, or both. `ndb_redo_log_reader` is intended for use primarily by NDB Cluster developers and Support personnel in debugging and diagnosing problems.

This utility remains under development, and its syntax and behavior are subject to change in future NDB Cluster releases.

The C++ source files for `ndb_redo_log_reader` can be found in the directory `/storage/ndb/src/kernel/blocks/dblqh/redoLogReader`.

Options that can be used with `ndb_redo_log_reader` are shown in the following table. Additional descriptions follow the table.

**Table 23.41 Command-line options used with the program ndb_redo_log_reader**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-dump` | Print dump info | (Supported in all NDB releases based on MySQL 8.0) |
| `--file-key=key`, `-K key` | Supply decryption key | ADDED: NDB 8.0.31 |
| `--file-key-from-stdin` | Supply decryption key using stdin | ADDED: NDB 8.0.31 |
| `-filedescriptors` | Print file descriptors only | (Supported in all NDB releases based on MySQL 8.0) |
| `--help` | Print usage information (has no short form) | (Supported in all NDB releases based on MySQL 8.0) |
| `-lap` | Provide lap info, with max GCI started and completed | (Supported in all NDB releases based on MySQL 8.0) |
| `-mbyte #` | Starting megabyte | (Supported in all NDB releases based on MySQL 8.0) |
| `-mbyteheaders` | Show only first page header of each megabyte in file | (Supported in all NDB releases based on MySQL 8.0) |
| `-nocheck` | Do not check records for errors | (Supported in all NDB releases based on MySQL 8.0) |
| `-noprint` | Do not print records | (Supported in all NDB releases based on MySQL 8.0) |
| `-page #` | Start with this page | (Supported in all NDB releases based on MySQL 8.0) |
| `-pageheaders` | Show page headers only | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-pageindex #` | Start with this page index | (Supported in all NDB releases based on MySQL 8.0) |
| `-twiddle` | Bit-shifted dump | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
ndb_redo_log_reader file_name [options]
```

*`file_name`* is the name of a cluster redo log file. redo log files are located in the numbered directories under the data node's data directory (`DataDir`); the path under this directory to the redo log files matches the pattern `ndb_nodeid_fs/D#/DBLQH/S#.FragLog`. *`nodeid`* is the data node's node ID. The two instances of `#` each represent a number (not necessarily the same number); the number following `D` is in the range 8-39 inclusive; the range of the number following `S` varies according to the value of the `NoOfFragmentLogFiles` configuration parameter, whose default value is 16; thus, the default range of the number in the file name is 0-15 inclusive. For more information, see NDB Cluster Data Node File System Directory.

The name of the file to be read may be followed by one or more of the options listed here:

- `-dump`

  | Command-Line Format | `-dump` |
  |---|---|

  Print dump info.

- `--file-key`, `-K`

  | Command-Line Format | `--file-key=key` |
  |---|---|
  | Introduced | 8.0.31-ndb-8.0.31 |

  Supply file decryption key using `stdin`, `tty`, or a `my.cnf` file.

- `--file-key-from-stdin`

  | Command-Line Format | `--file-key-from-stdin` |
  |---|---|
  | Introduced | 8.0.31-ndb-8.0.31 |

  Supply file decryption key using `stdin`.

- | Command-Line Format | `-filedescriptors` |
  |---|---|

  `-filedescriptors`: Print file descriptors only.

- | Command-Line Format | `--help` |
  |---|---|

  `--help`: Print usage information.

- `-lap`

  | Command-Line Format | `-lap` |
  |---|---|

  Provide lap info, with max GCI started and completed.

- | Command-Line Format | `-mbyte #` |
  |---|---|
  | Type | Numeric |

| | |
|---|---|
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `15` |

`-mbyte #`: Starting megabyte.

`#` is an integer in the range 0 to 15, inclusive.

- | | |
|---|---|
| Command-Line Format | `-mbyteheaders` |

`-mbyteheaders`: Show only the first page header of every megabyte in the file.

- | | |
|---|---|
| Command-Line Format | `-noprint` |

`-noprint`: Do not print the contents of the log file.

- | | |
|---|---|
| Command-Line Format | `-nocheck` |

`-nocheck`: Do not check the log file for errors.

- | | |
|---|---|
| Command-Line Format | `-page #` |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `31` |

`-page #`: Start at this page.

`#` is an integer in the range 0 to 31, inclusive.

- | | |
|---|---|
| Command-Line Format | `-pageheaders` |

`-pageheaders`: Show page headers only.

- | | |
|---|---|
| Command-Line Format | `-pageindex #` |
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `12` |
| Maximum Value | `8191` |

`-pageindex #`: Start at this page index.

`#` is an integer between 12 and 8191, inclusive.

- `-twiddle`

| | |
|---|---|
| Command-Line Format | `-twiddle` |

Bit-shifted dump.

Like `ndb_print_backup_file` and `ndb_print_schema_file` (and unlike most of the `NDB` utilities that are intended to be run on a management server host or to connect to a management server) `ndb_redo_log_reader` must be run on a cluster data node, since it accesses the data node file system directly. Because it does not make use of the management server, this utility can be used when the management server is not running, and even when the cluster has been completely shut down.

## 23.5.23 ndb_restore — Restore an NDB Cluster Backup

The NDB Cluster restoration program is implemented as a separate command-line utility `ndb_restore`, which can normally be found in the MySQL `bin` directory. This program reads the files created as a result of the backup and inserts the stored information into the database.

In NDB 7.6 and earlier, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency has been removed in NDB 8.0, eliminating the extraneous output.

`ndb_restore` must be executed once for each of the backup files that were created by the `START BACKUP` command used to create the backup (see Section 23.6.8.2, "Using The NDB Cluster Management Client to Create a Backup"). This is equal to the number of data nodes in the cluster at the time that the backup was created.

**Note**

Before using `ndb_restore`, it is recommended that the cluster be running in single user mode, unless you are restoring multiple data nodes in parallel. See Section 23.6.6, "NDB Cluster Single User Mode", for more information.

Options that can be used with `ndb_restore` are shown in the following table. Additional descriptions follow the table.

**Table 23.42 Command-line options used with the program ndb_restore**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--allow-pk-changes[=0\|1]` | Allow changes to set of columns making up table's primary key | ADDED: NDB 8.0.21 |
| `--append` | Append data to tab-delimited file | (Supported in all NDB releases based on MySQL 8.0) |
| `--backup-password=password` | Supply a password for decrypting an encrypted backup with --decrypt; see documentation for allowed values | ADDED: NDB 8.0.22 |
| `--backup-password-from-stdin` | Get decryption password in a secure fashion from STDIN; use together with --decrypt option | ADDED: NDB 8.0.24 |
| `--backup-path=path` | Path to backup files directory | (Supported in all NDB releases based on MySQL 8.0) |
| `--backupid=#`, `-b #` | Restore from backup having this ID | (Supported in all NDB releases based on MySQL 8.0) |
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect=connection_string`, `-c connection_string` | Alias for --connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| -c connection_string | | |
| --core-file | Write core file on error; used in debugging | (Supported in all NDB releases based on MySQL 8.0) |
| --decrypt | Decrypt an encrypted backup; requires --backup-password | ADDED: NDB 8.0.22 |
| --defaults-extra-file=path | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| --defaults-file=path | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| --defaults-group-suffix=string | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| --disable-indexes | Causes indexes from backup to be ignored; may decrease time needed to restore data | (Supported in all NDB releases based on MySQL 8.0) |
| --dont-ignore-systab-0, -f | Do not ignore system table during restore; experimental only; not for production use | (Supported in all NDB releases based on MySQL 8.0) |
| --exclude-databases=list | List of one or more databases to exclude (includes those not named) | (Supported in all NDB releases based on MySQL 8.0) |
| --exclude-intermediate-sql-tables[=TRUE\|FALSE] | Do not restore any intermediate tables (having names prefixed with '#sql-') that were left over from copying ALTER TABLE operations; specify FALSE to restore such tables | (Supported in all NDB releases based on MySQL 8.0) |
| --exclude-missing-columns | Causes columns from backup version of table that are missing from version of table in database to be ignored | (Supported in all NDB releases based on MySQL 8.0) |
| --exclude-missing-tables | Causes tables from backup that are missing from database to be ignored | (Supported in all NDB releases based on MySQL 8.0) |
| --exclude-tables=list | List of one or more tables to exclude (includes those in same database that are not named); each table reference must include database name | (Supported in all NDB releases based on MySQL 8.0) |
| --fields-enclosed-by=char | Fields are enclosed by this character | (Supported in all NDB releases based on MySQL 8.0) |
| --fields-optionally-enclosed-by | Fields are optionally enclosed by this character | (Supported in all NDB releases based on MySQL 8.0) |
| --fields-terminated-by=char | Fields are terminated by this character | (Supported in all NDB releases based on MySQL 8.0) |
| --help, -? | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| --hex | Print binary types in hexadecimal format | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--ignore-extended-pk-updates[=0\|1]` | Ignore log entries containing updates to columns now included in extended primary key | ADDED: NDB 8.0.21 |
| `--include-databases=list` | List of one or more databases to restore (excludes those not named) | (Supported in all NDB releases based on MySQL 8.0) |
| `--include-stored-grants` | Restore shared users and grants to ndb_sql_metadata table | ADDED: NDB 8.0.19 |
| `--include-tables=list` | List of one or more tables to restore (excludes those in same database that are not named); each table reference must include database name | (Supported in all NDB releases based on MySQL 8.0) |
| `--lines-terminated-by=char` | Lines are terminated by this character | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--lossy-conversions`, `-L` | Allow lossy conversions of column values (type demotions or changes in sign) when restoring data from backup | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-binlog` | If mysqld is connected and using binary logging, do not log restored data | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-restore-disk-objects`, `-d` | Do not restore objects relating to Disk Data | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-upgrade`, `-u` | Do not upgrade array type for varsize attributes which do not already resize VAR data, and do not change column attributes | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodegroup-map=map`, `-z` | Specify node group map; unused, unsupported | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--nodeid=#`, `-n #` | ID of node where backup was taken | (Supported in all NDB releases based on MySQL 8.0) |
| `--num-slices=#` | Number of slices to apply when restoring by slice | ADDED: NDB 8.0.20 |
| `--parallelism=#`, `-p #` | Number of parallel transactions to use while restoring data | (Supported in all NDB releases based on MySQL 8.0) |
| `--preserve-trailing-spaces`, `-P` | Allow preservation of trailing spaces (including padding) when promoting fixed-width string types to variable-width types | (Supported in all NDB releases based on MySQL 8.0) |
| `--print` | Print metadata, data, and log to stdout (equivalent to --print-meta --print-data --print-log) | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-data` | Print data to stdout | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-log` | Print log to stdout | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-meta` | Print metadata to stdout | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-sql-log` | Write SQL log to stdout | (Supported in all NDB releases based on MySQL 8.0) |
| `--progress-frequency=#` | Print status of restore each given number of seconds | (Supported in all NDB releases based on MySQL 8.0) |
| `--promote-attributes`, `-A` | Allow attributes to be promoted when restoring data from backup | (Supported in all NDB releases based on MySQL 8.0) |
| `--rebuild-indexes` | Causes multithreaded rebuilding of ordered indexes found in backup; number of threads used is determined by setting BuildIndexThreads | (Supported in all NDB releases based on MySQL 8.0) |
| `--remap-column=string` | Apply offset to value of specified column using indicated function and arguments. Format is [db].[tbl].[col]:[fn]:[args]; see documentation for details | ADDED: NDB 8.0.21 |
| `--restore-data`, | Restore table data and logs into NDB Cluster using NDB API | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-r` | | |
| `--restore-epoch`, `-e` | Restore epoch info into status table; useful on replica cluster for starting replication; updates or inserts row in mysql.ndb_apply_status with ID 0 | (Supported in all NDB releases based on MySQL 8.0) |
| `--restore-meta`, `-m` | Restore metadata to NDB Cluster using NDB API | (Supported in all NDB releases based on MySQL 8.0) |
| `--restore-privilege-tables` | Restore MySQL privilege tables that were previously moved to NDB | DEPRECATED: NDB 8.0.16 |
| `--rewrite-database=string` | Restore to differently named database; format is olddb,newdb | (Supported in all NDB releases based on MySQL 8.0) |
| `--skip-broken-objects` | Ignore missing blob tables in backup file | (Supported in all NDB releases based on MySQL 8.0) |
| `--skip-table-check`, `-s` | Skip table structure check during restore | (Supported in all NDB releases based on MySQL 8.0) |
| `--skip-unknown-objects` | Causes schema objects not recognized by ndb_restore to be ignored when restoring backup made from newer NDB version to older version | (Supported in all NDB releases based on MySQL 8.0) |
| `--slice-id=#` | Slice ID, when restoring by slices | ADDED: NDB 8.0.20 |
| `--tab=path`, `-T path` | Creates a tab-separated .txt file for each table in path provided | (Supported in all NDB releases based on MySQL 8.0) |
| `--timestamp-printouts{=true\|false}` | Prefix all info, error, and debug log messages with timestamps | ADDED: NDB 8.0.33 |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--verbose=#` | Level of verbosity in output | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--with-apply-status` | Restore the ndb_apply_status table. Requires --restore-data | ADDED: NDB 8.0.29 |

- `--allow-pk-changes`

| | |
|---|---|
| Command-Line Format | `--allow-pk-changes[=0\|1]` |
| Introduced | 8.0.21-ndb-8.0.21 |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |

| | |
|---|---|
| Maximum Value | 1 |

When this option is set to 1, ndb_restore allows the primary keys in a table definition to differ from that of the same table in the backup. This may be desirable when backing up and restoring between different schema versions with primary key changes on one or more tables, and it appears that performing the restore operation using ndb_restore is simpler or more efficient than issuing many ALTER TABLE statements after restoring table schemas and data.

The following changes in primary key definitions are supported by --allow-pk-changes:

- **Extending the primary key**: A non-nullable column that exists in the table schema in the backup becomes part of the table's primary key in the database.

**Important**

When extending a table's primary key, any columns which become part of primary key must not be updated while the backup is being taken; any such updates discovered by ndb_restore cause the restore operation to fail, even when no change in value takes place. In some cases, it may be possible to override this behavior using the --ignore-extended-pk-updates option; see the description of this option for more information.

- **Contracting the primary key (1)**: A column that is already part of the table's primary key in the backup schema is no longer part of the primary key, but remains in the table.
- **Contracting the primary key (2)**: A column that is already part of the table's primary key in the backup schema is removed from the table entirely.

These differences can be combined with other schema differences supported by ndb_restore, including changes to blob and text columns requiring the use of staging tables.

Basic steps in a typical scenario using primary key schema changes are listed here:

1. Restore table schemas using ndb_restore --restore-meta
2. Alter schema to that desired, or create it
3. Back up the desired schema
4. Run ndb_restore --disable-indexes using the backup from the previous step, to drop indexes and constraints
5. Run ndb_restore --allow-pk-changes (possibly along with --ignore-extended-pk-updates, --disable-indexes, and possibly other options as needed) to restore all data
6. Run ndb_restore --rebuild-indexes using the backup made with the desired schema, to rebuild indexes and constraints

When extending the primary key, it may be necessary for ndb_restore to use a temporary secondary unique index during the restore operation to map from the old primary key to the new one. Such an index is created only when necessary to apply events from the backup log to a table which has an extended primary key. This index is named NDB$RESTORE_PK_MAPPING, and is created on each table requiring it; it can be shared, if necessary, by multiple instances of ndb_restore instances running in parallel. (Running ndb_restore --rebuild-indexes at the end of the restore process causes this index to be dropped.)

- `--append`

| Command-Line Format | `--append` |
|---|---|

When used with the `--tab` and `--print-data` options, this causes the data to be appended to any existing files having the same names.

- `--backup-path=dir_name`

| Command-Line Format | `--backup-path=path` |
|---|---|
| Type | Directory name |
| Default Value | `./` |

The path to the backup directory is required; this is supplied to `ndb_restore` using the `--backup-path` option, and must include the subdirectory corresponding to the ID backup of the backup to be restored. For example, if the data node's `DataDir` is `/var/lib/mysql-cluster`, then the backup directory is `/var/lib/mysql-cluster/BACKUP`, and the backup files for the backup with the ID 3 can be found in `/var/lib/mysql-cluster/BACKUP/BACKUP-3`. The path may be absolute or relative to the directory in which the `ndb_restore` executable is located, and may be optionally prefixed with `backup-path=`.

It is possible to restore a backup to a database with a different configuration than it was created from. For example, suppose that a backup with backup ID `12`, created in a cluster with two storage nodes having the node IDs `2` and `3`, is to be restored to a cluster with four nodes. Then `ndb_restore` must be run twice—once for each storage node in the cluster where the backup was taken. However, `ndb_restore` cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version. See Section 23.3.7, “Upgrading and Downgrading NDB Cluster”, for more information.

**Important**

It is not possible to restore a backup made from a newer version of NDB Cluster using an older version of `ndb_restore`. You can restore a backup made from a newer version of MySQL to an older cluster, but you must use a copy of `ndb_restore` from the newer NDB Cluster version to do so.

For example, to restore a cluster backup taken from a cluster running NDB Cluster 8.0.34 to a cluster running NDB Cluster 7.6.27, you must use the `ndb_restore` that comes with the NDB Cluster 7.6.27 distribution.

For more rapid restoration, the data may be restored in parallel, provided that there is a sufficient number of cluster connections available. That is, when restoring to multiple nodes in parallel, you must have an `[api]` or `[mysqld]` section in the cluster `config.ini` file available for each concurrent `ndb_restore` process. However, the data files must always be applied before the logs.

- `--backup-password=password`

| Command-Line Format | `--backup-password=password` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |
| Type | String |
| Default Value | `[none]` |

This option specifies a password to be used when decrypting an encrypted backup with the `--decrypt` option. This must be the same password that was used to encrypt the backup.

The password must be 1 to 256 characters in length, and must be enclosed by single or double quotation marks. It can contain any of the ASCII characters having character codes 32, 35, 38,

40-91, 93, 95, and 97-126; in other words, it can use any printable ASCII characters except for `!`, `'`, `"`, `$`, `%`, `\`, and `^`.

In MySQL 8.0.24 and later, it is possible to omit the password, in which case `ndb_restore` waits for it to be supplied from `stdin`, as when using `--backup-password-from-stdin`.

- `--backup-password-from-stdin[=TRUE|FALSE]`

| Command-Line Format | `--backup-password-from-stdin` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |

When used in place of `--backup-password`, this option enables input of the backup password from the system shell (`stdin`), similar to how this is done when supplying the password interactively to `mysql` when using the `--password` without supplying the password on the command line.

- `--backupid=#`, `-b`

| Command-Line Format | `--backupid=#` |
| --- | --- |
| Type | Numeric |
| Default Value | `none` |

This option is used to specify the ID or sequence number of the backup, and is the same number shown by the management client in the `Backup backup_id completed` message displayed upon completion of a backup. (See Section 23.6.8.2, “Using The NDB Cluster Management Client to Create a Backup”.)

**Important**

When restoring cluster backups, you must be sure to restore all data nodes from backups having the same backup ID. Using files from different backups results at best in restoring the cluster to an inconsistent state, and is likely to fail altogether.

In NDB 8.0, this option is required.

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
| --- | --- |
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect`, `-c`

| Command-Line Format | `--connect=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `localhost:1186` |

Alias for `--ndb-connectstring`.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
| --- | --- |
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |

| | |
|---|---|
| Maximum Value | 12 |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| | |
|---|---|
| Command-Line Format | `--connect-retry-delay=#` |
| Type | Integer |
| Default Value | 5 |
| Minimum Value | 0 |
| Maximum Value | 5 |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| | |
|---|---|
| Command-Line Format | `--connect-string=connection_string` |
| Type | String |
| Default Value | [none] |

Same as `--ndb-connectstring`.

- `--core-file`

| | |
|---|---|
| Command-Line Format | `--core-file` |

Write core file on error; used in debugging.

- `--decrypt`

| | |
|---|---|
| Command-Line Format | `--decrypt` |
| Introduced | 8.0.22-ndb-8.0.22 |

Decrypt an encrypted backup using the password supplied by the `--backup-password` option.

- `--defaults-extra-file`

| | |
|---|---|
| Command-Line Format | `--defaults-extra-file=path` |
| Type | String |
| Default Value | [none] |

Read given file after global files are read.

- `--defaults-file`

| | |
|---|---|
| Command-Line Format | `--defaults-file=path` |
| Type | String |
| Default Value | [none] |

Read default options from given file only.

- `--defaults-group-suffix`

| | |
|---|---|
| Command-Line Format | `--defaults-group-suffix=string` |
| Type | String |

| Default Value | [none] |
|---|---|

Also read groups with concat(group, suffix).

- `--disable-indexes`

| Command-Line Format | --disable-indexes |
|---|---|

Disable restoration of indexes during restoration of the data from a native NDB backup. Afterwards, you can restore indexes for all tables at once with multithreaded building of indexes using `--rebuild-indexes`, which should be faster than rebuilding indexes concurrently for very large tables.

In NDB 8.0.27 and later, this option also drops any foreign keys specified in the backup.

Prior to NDB 8.0.29, attempting to access from MySQL an NDB table for which one or more indexes could not be found was always rejected with error `4243 Index not found`. Beginning with NDB 8.0.29, it is possible for MySQL to open such a table, provided the query does not use any of the affected indexes; otherwise the query is rejected with ER_NOT_KEYFILE. In the latter case, you can temporarily work around the problem by executing an `ALTER TABLE` statement such as this one:

```
ALTER TABLE tbl ALTER INDEX idx INVISIBLE;
```

This causes MySQL to ignore the index `idx` on table `tbl`. See Primary Keys and Indexes, for more information.

- `--dont-ignore-systab-0`, `-f`

| Command-Line Format | --dont-ignore-systab-0 |
|---|---|

Normally, when restoring table data and metadata, `ndb_restore` ignores the copy of the NDB system table that is present in the backup. `--dont-ignore-systab-0` causes the system table to be restored. *This option is intended for experimental and development use only, and is not recommended in a production environment*.

- `--exclude-databases=db-list`

| Command-Line Format | --exclude-databases=list |
|---|---|
| Type | String |
| Default Value | |

Comma-delimited list of one or more databases which should not be restored.

This option is often used in combination with `--exclude-tables`; see that option's description for further information and examples.

- `--exclude-intermediate-sql-tables[=TRUE|FALSE]`

| Command-Line Format | --exclude-intermediate-sql-tables[=TRUE|FALSE] |
|---|---|
| Type | Boolean |
| Default Value | TRUE |

When performing copying `ALTER TABLE` operations, `mysqld` creates intermediate tables (whose names are prefixed with `#sql-`). When `TRUE`, the `--exclude-intermediate-sql-tables` option keeps `ndb_restore` from restoring such tables that may have been left over from these 4437 operations. This option is `TRUE` by default.

- `--exclude-missing-columns`

| Command-Line Format | `--exclude-missing-columns` |
|---|---|

It is possible to restore only selected table columns using this option, which causes `ndb_restore` to ignore any columns missing from tables being restored as compared to the versions of those tables found in the backup. This option applies to all tables being restored. If you wish to apply this option only to selected tables or databases, you can use it in combination with one or more of the `--include-*` or `--exclude-*` options described elsewhere in this section to do so, then restore data to the remaining tables using a complementary set of these options.

- `--exclude-missing-tables`

| Command-Line Format | `--exclude-missing-tables` |
|---|---|

It is possible to restore only selected tables using this option, which causes `ndb_restore` to ignore any tables from the backup that are not found in the target database.

- `--exclude-tables=`*`table-list`*

| Command-Line Format | `--exclude-tables=list` |
|---|---|
| Type | String |
| Default Value | |

List of one or more tables to exclude; each table reference must include the database name. Often used together with `--exclude-databases`.

When `--exclude-databases` or `--exclude-tables` is used, only those databases or tables named by the option are excluded; all other databases and tables are restored by `ndb_restore`.

This table shows several invocations of `ndb_restore` using `--exclude-*` options (other options possibly required have been omitted for clarity), and the effects these options have on restoring from an NDB Cluster backup:

**Table 23.43 Several invocations of ndb_restore using --exclude-* options, and the effects these options have on restoring from an NDB Cluster backup.**

| Option | Result |
|---|---|
| `--exclude-databases=db1` | All tables in all databases except `db1` are restored; no tables in `db1` are restored |
| `--exclude-databases=db1,db2` (or `--exclude-databases=db1 --exclude-databases=db2`) | All tables in all databases except `db1` and `db2` are restored; no tables in `db1` or `db2` are restored |
| `--exclude-tables=db1.t1` | All tables except `t1` in database `db1` are restored; all other tables in `db1` are restored; all tables in all other databases are restored |
| `--exclude-tables=db1.t2,db2.t1` (or `--exclude-tables=db1.t2 --exclude-tables=db2.t1)` | All tables in database `db1` except for `t2` and all tables in database `db2` except for table `t1` are restored; no other tables in `db1` or `db2` are |

| Option | Result |
|---|---|
| | restored; all tables in all other databases are restored |

You can use these two options together. For example, the following causes all tables in all databases *except for* databases `db1` and `db2`, and tables `t1` and `t2` in database `db3`, to be restored:

```
$> ndb_restore [...] --exclude-databases=db1,db2 --exclude-tables=db3.t1,db3.t2
```

(Again, we have omitted other possibly necessary options in the interest of clarity and brevity from the example just shown.)

You can use `--include-*` and `--exclude-*` options together, subject to the following rules:

- The actions of all `--include-*` and `--exclude-*` options are cumulative.
- All `--include-*` and `--exclude-*` options are evaluated in the order passed to ndb_restore, from right to left.
- In the event of conflicting options, the first (rightmost) option takes precedence. In other words, the first option (going from right to left) that matches against a given database or table “wins”.

For example, the following set of options causes `ndb_restore` to restore all tables from database `db1` except `db1.t1`, while restoring no other tables from any other databases:

```
--include-databases=db1 --exclude-tables=db1.t1
```

However, reversing the order of the options just given simply causes all tables from database `db1` to be restored (including `db1.t1`, but no tables from any other database), because the `--include-databases` option, being farthest to the right, is the first match against database `db1` and thus takes precedence over any other option that matches `db1` or any tables in `db1`:

```
--exclude-tables=db1.t1 --include-databases=db1
```

- `--fields-enclosed-by`=*`char`*

| Command-Line Format | `--fields-enclosed-by=char` |
|---|---|
| Type | String |
| Default Value | |

Each column value is enclosed by the string passed to this option (regardless of data type; see the description of `--fields-optionally-enclosed-by`).

- `--fields-optionally-enclosed-by`

| Command-Line Format | `--fields-optionally-enclosed-by` |
|---|---|
| Type | String |
| Default Value | |

The string passed to this option is used to enclose column values containing character data (such as `CHAR`, `VARCHAR`, `BINARY`, `TEXT`, or `ENUM`).

- `--fields-terminated-by`=*`char`*

| Command-Line Format | `--fields-terminated-by=char` |
|---|---|
| Type | String |

| Default Value | `\t (tab)` |
|---|---|

The string passed to this option is used to separate column values. The default value is a tab character (`\t`).

- `--help`

| Command-Line Format | `--help` |
|---|---|

Display help text and exit.

- `--hex`

| Command-Line Format | `--hex` |
|---|---|

If this option is used, all binary values are output in hexadecimal format.

- `--ignore-extended-pk-updates`

| Command-Line Format | `--ignore-extended-pk-updates[=0\|1]` |
|---|---|
| Introduced | 8.0.21-ndb-8.0.21 |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `1` |

When using `--allow-pk-changes`, columns which become part of a table's primary key must not be updated while the backup is being taken; such columns should keep the same values from the time values are inserted into them until the rows containing the values are deleted. If `ndb_restore` encounters updates to these columns when restoring a backup, the restore fails. Because some applications may set values for all columns when updating a row, even when some column values are not changed, the backup may include log events appearing to update columns which are not in fact modified. In such cases you can set `--ignore-extended-pk-updates` to `1`, forcing `ndb_restore` to ignore such updates.

**Important**

When causing these updates to be ignored, the user is responsible for ensuring that there are no updates to the values of any columns that become part of the primary key.

For more information, see the description of `--allow-pk-changes`.

- `--include-databases=`*`db-list`*

| Command-Line Format | `--include-databases=list` |
|---|---|
| Type | String |
| Default Value | |

Comma-delimited list of one or more databases to restore. Often used together with `--include-tables`; see the description of that option for further information and examples.

- `--include-stored-grants`

| Command-Line Format | `--include-stored-grants` |
|---|---|
| Introduced | 8.0.19-ndb-8.0.19 |

In NDB 8.0, `ndb_restore` does not by default restore shared users and grants (see Section 23.6.13, “Privilege Synchronization and NDB_STORED_USER”) to the `ndb_sql_metadata` table. Specifying this option causes it to do so.

- `--include-tables=table-list`

| Command-Line Format | `--include-tables=list` |
| --- | --- |
| Type | String |
| Default Value | |

Comma-delimited list of tables to restore; each table reference must include the database name.

When `--include-databases` or `--include-tables` is used, only those databases or tables named by the option are restored; all other databases and tables are excluded by `ndb_restore`, and are not restored.

The following table shows several invocations of `ndb_restore` using `--include-*` options (other options possibly required have been omitted for clarity), and the effects these have on restoring from an NDB Cluster backup:

**Table 23.44 Several invocations of ndb_restore using --include-* options, and their effects on restoring from an NDB Cluster backup.**

| Option | Result |
| --- | --- |
| `--include-databases=db1` | Only tables in database `db1` are restored; all tables in all other databases are ignored |
| `--include-databases=db1,db2` (or `--include-databases=db1 --include-databases=db2`) | Only tables in databases `db1` and `db2` are restored; all tables in all other databases are ignored |
| `--include-tables=db1.t1` | Only table `t1` in database `db1` is restored; no other tables in `db1` or in any other database are restored |
| `--include-tables=db1.t2,db2.t1` (or `--include-tables=db1.t2 --include-tables=db2.t1`) | Only the table `t2` in database `db1` and the table `t1` in database `db2` are restored; no other tables in `db1`, `db2`, or any other database are restored |

You can also use these two options together. For example, the following causes all tables in databases `db1` and `db2`, together with the tables `t1` and `t2` in database `db3`, to be restored (and no other databases or tables):

```
$> ndb_restore [...] --include-databases=db1,db2 --include-tables=db3.t1,db3.t2
```

(Again we have omitted other, possibly required, options in the example just shown.)

It also possible to restore only selected databases, or selected tables from a single database, without any `--include-*` (or `--exclude-*`) options, using the syntax shown here:

```
ndb_restore other_options db_name,[db_name[,...] | tbl_name[,tbl_name][,...]]
```

In other words, you can specify either of the following to be restored:

- All tables from one or more databases
- One or more tables from a single database

- `--lines-terminated-by=char`

| Command-Line Format | `--lines-terminated-by=char` |
| --- | --- |
| Type | String |
| Default Value | `\n (linebreak)` |

Specifies the string used to end each line of output. The default is a linefeed character (`\n`).

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--lossy-conversions`, `-L`

| Command-Line Format | `--lossy-conversions` |
| --- | --- |

This option is intended to complement the `--promote-attributes` option. Using `--lossy-conversions` allows lossy conversions of column values (type demotions or changes in sign) when restoring data from backup. With some exceptions, the rules governing demotion are the same as for MySQL replication; see Replication of Columns Having Different Data Types, for information about specific type conversions currently supported by attribute demotion.

Beginning with NDB 8.0.26, this option also makes it possible to restore a `NULL` column as `NOT NULL`. The column must not contain any `NULL` entries; otherwise `ndb_restore` stops with an error.

`ndb_restore` reports any truncation of data that it performs during lossy conversions once per attribute and column.

- `--no-binlog`

| Command-Line Format | `--no-binlog` |
| --- | --- |

This option prevents any connected SQL nodes from writing data restored by `ndb_restore` to their binary logs.

- `--no-restore-disk-objects`, `-d`

| Command-Line Format | `--no-restore-disk-objects` |
| --- | --- |

This option stops `ndb_restore` from restoring any NDB Cluster Disk Data objects, such as tablespaces and log file groups; see Section 23.6.11, “NDB Cluster Disk Data Tables”, for more information about these.

- `--no-upgrade`, `-u`

| Command-Line Format | `--no-upgrade` |
| --- | --- |

When using `ndb_restore` to restore a backup, `VARCHAR` columns created using the old fixed format are resized and recreated using the variable-width format now employed. This behavior can be overridden by specifying `--no-upgrade`.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-nodegroup-map=map`, `-z`

| Command-Line Format | `--ndb-nodegroup-map=map` |
| --- | --- |

Intended for restoring a backup taken from one node group to a different node group, but never completely implemented; unsupported.

All code supporting this option was removed in NDB 8.0.27; in this and later versions, any value set for it is ignored, and the option itself does nothing.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--nodeid=#`, `-n`

| Command-Line Format | `--nodeid=#` |
| --- | --- |
| Type | Numeric |

| Default Value | none |
|---|---|

Specify the node ID of the data node on which the backup was taken.

When restoring to a cluster with different number of data nodes from that where the backup was taken, this information helps identify the correct set or sets of files to be restored to a given node. (In such cases, multiple files usually need to be restored to a single data node.) See Section 23.5.23.2, “Restoring to a different number of data nodes”, for additional information and examples.

In NDB 8.0, this option is required.

- `--num-slices=#`

| Command-Line Format | --num-slices=# |
|---|---|
| Introduced | 8.0.20-ndb-8.0.20 |
| Type | Integer |
| Default Value | 1 |
| Minimum Value | 1 |
| Maximum Value | 1024 |

When restoring a backup by slices, this option sets the number of slices into which to divide the backup. This allows multiple instances of `ndb_restore` to restore disjoint subsets in parallel, potentially reducing the amount of time required to perform the restore operation.

A *slice* is a subset of the data in a given backup; that is, it is a set of fragments having the same slice ID, specified using the `--slice-id` option. The two options must always be used together, and the value set by `--slice-id` must always be less than the number of slices.

`ndb_restore` encounters fragments and assigns each one a fragment counter. When restoring by slices, a slice ID is assigned to each fragment; this slice ID is in the range 0 to 1 less than the number of slices. For a table that is not a `BLOB` table, the slice to which a given fragment belongs is determined using the formula shown here:

```
[slice_ID] = [fragment_counter] % [number_of_slices]
```

For a `BLOB` table, a fragment counter is not used; the fragment number is used instead, along with the ID of the main table for the `BLOB` table (recall that `NDB` stores *`BLOB`* values in a separate table internally). In this case, the slice ID for a given fragment is calculated as shown here:

```
[slice_ID] =
([main_table_ID] + [fragment_ID]) % [number_of_slices]
```

Thus, restoring by *`N`* slices means running *`N`* instances of `ndb_restore`, all with `--num-slices=N` (along with any other necessary options) and one each with `--slice-id=1`, `--slice-id=2`, `--slice-id=3`, and so on through `slice-id=N-1`.

**Example.** Assume that you want to restore a backup named `BACKUP-1`, found in the default directory `/var/lib/mysql-cluster/BACKUP/BACKUP-3` on the node file system on each data node, to a cluster with four data nodes having the node IDs 1, 2, 3, and 4. To perform this operation using five slices, execute the sets of commands shown in the following list:

1. Restore the cluster metadata using `ndb_restore` as shown here:

   ```
   $> ndb_restore -b 1 -n 1 -m --disable-indexes --backup-path=/home/ndbuser/backups
   ```

2. Restore the cluster data to the data nodes invoking `ndb_restore` as shown here:

   ```
   $> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKUP/B
   $> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKUP/B
   $> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKUP/B
   ```

```
$> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 1 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKU

$> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 2 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKU

$> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 3 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKU

$> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=0 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=1 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=2 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=3 --backup-path=/var/lib/mysql-cluster/BACKU
$> ndb_restore -b 1 -n 4 -r --num-slices=5 --slice-id=4 --backup-path=/var/lib/mysql-cluster/BACKU
```

All of the commands just shown in this step can be executed in parallel, provided there are enough slots for connections to the cluster (see the description for the `--backup-path` option).

3. Restore indexes as usual, as shown here:

```
$> ndb_restore -b 1 -n 1 --rebuild-indexes --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
```

4. Finally, restore the epoch, using the command shown here:

```
$> ndb_restore -b 1 -n 1 --restore-epoch --backup-path=/var/lib/mysql-cluster/BACKUP/BACKUP-1
```

You should use slicing to restore the cluster data only; it is not necessary to employ `--num-slices` or `--slice-id` when restoring the metadata, indexes, or epoch information. If either or both of these options are used with the `ndb_restore` options controlling restoration of these, the program ignores them.

The effects of using the `--parallelism` option on the speed of restoration are independent of those produced by slicing or parallel restoration using multiple instances of `ndb_restore` (`--parallelism` specifies the number of parallel transactions executed by a *single* `ndb_restore` thread), but it can be used together with either or both of these. You should be aware that increasing `--parallelism` causes `ndb_restore` to impose a greater load on the cluster; if the system can handle this, restoration should complete even more quickly.

The value of `--num-slices` is not directly dependent on values relating to hardware such as number of CPUs or CPU cores, amount of RAM, and so forth, nor does it depend on the number of LDMs.

It is possible to employ different values for this option on different data nodes as part of the same restoration; doing so should not in and of itself produce any ill effects.

- `--parallelism=#`, `-p`

| Command-Line Format | `--parallelism=#` |
|---|---|
| Type | Numeric |
| Default Value | `128` |
| Minimum Value | `1` |

| Maximum Value | 1024 |
|---|---|

ndb_restore uses single-row transactions to apply many rows concurrently. This parameter determines the number of parallel transactions (concurrent rows) that an instance of ndb_restore tries to use. By default, this is 128; the minimum is 1, and the maximum is 1024.

The work of performing the inserts is parallelized across the threads in the data nodes involved. This mechanism is employed for restoring bulk data from the .Data file—that is, the fuzzy snapshot of the data; it is not used for building or rebuilding indexes. The change log is applied serially; index drops and builds are DDL operations and handled separately. There is no thread-level parallelism on the client side of the restore.

- --preserve-trailing-spaces, -P

| Command-Line Format | --preserve-trailing-spaces |
|---|---|

Cause trailing spaces to be preserved when promoting a fixed-width character data type to its variable-width equivalent—that is, when promoting a CHAR column value to VARCHAR, or a BINARY column value to VARBINARY. Otherwise, any trailing spaces are dropped from such column values when they are inserted into the new columns.

**Note**

Although you can promote CHAR columns to VARCHAR and BINARY columns to VARBINARY, you cannot promote VARCHAR columns to CHAR or VARBINARY columns to BINARY.

- --print

| Command-Line Format | --print |
|---|---|

Causes ndb_restore to print all data, metadata, and logs to stdout. Equivalent to using the --print-data, --print-meta, and --print-log options together.

**Note**

Use of --print or any of the --print_* options is in effect performing a dry run. Including one or more of these options causes any output to be redirected to stdout; in such cases, ndb_restore makes no attempt to restore data or metadata to an NDB Cluster.

- --print-data

| Command-Line Format | --print-data |
|---|---|

Cause ndb_restore to direct its output to stdout. Often used together with one or more of --tab, --fields-enclosed-by, --fields-optionally-enclosed-by, --fields-terminated-by, --hex, and --append.

TEXT and BLOB column values are always truncated. Such values are truncated to the first 256 bytes in the output. This cannot currently be overridden when using --print-data.

- --print-defaults

| Command-Line Format | --print-defaults |
|---|---|

Print program argument list and exit.

- `--print-log`

| Command-Line Format | `--print-log` |
|---|---|

Cause `ndb_restore` to output its log to `stdout`.

- `--print-meta`

| Command-Line Format | `--print-meta` |
|---|---|

Print all metadata to `stdout`.

- `print-sql-log`

| Command-Line Format | `--print-sql-log` |
|---|---|

Log SQL statements to `stdout`. Use the option to enable; normally this behavior is disabled. The option checks before attempting to log whether all the tables being restored have explicitly defined primary keys; queries on a table having only the hidden primary key implemented by `NDB` cannot be converted to valid SQL.

This option does not work with tables having `BLOB` columns.

- `--progress-frequency=`*`N`*

| Command-Line Format | `--progress-frequency=#` |
|---|---|
| Type | Numeric |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `65535` |

Print a status report each *`N`* seconds while the backup is in progress. 0 (the default) causes no status reports to be printed. The maximum is 65535.

- `--promote-attributes`, `-A`

| Command-Line Format | `--promote-attributes` |
|---|---|

`ndb_restore` supports limited *attribute promotion* in much the same way that it is supported by MySQL replication; that is, data backed up from a column of a given type can generally be restored to a column using a “larger, similar” type. For example, data from a `CHAR(20)` column can be restored to a column declared as `VARCHAR(20)`, `VARCHAR(30)`, or `CHAR(30)`; data from a `MEDIUMINT` column can be restored to a column of type `INT` or `BIGINT`. See Replication of Columns Having Different Data Types, for a table of type conversions currently supported by attribute promotion.

Beginning with NDB 8.0.26, this option also makes it possible to restore a `NOT NULL` column as `NULL`.

Attribute promotion by `ndb_restore` must be enabled explicitly, as follows:

1. Prepare the table to which the backup is to be restored. `ndb_restore` cannot be used to re-create the table with a different definition from the original; this means that you must either create

the table manually, or alter the columns which you wish to promote using `ALTER TABLE` after restoring the table metadata but before restoring the data.

2. Invoke `ndb_restore` with the `--promote-attributes` option (short form `-A`) when restoring the table data. Attribute promotion does not occur if this option is not used; instead, the restore operation fails with an error.

When converting between character data types and `TEXT` or `BLOB`, only conversions between character types (`CHAR` and `VARCHAR`) and binary types (`BINARY` and `VARBINARY`) can be performed at the same time. For example, you cannot promote an `INT` column to `BIGINT` while promoting a `VARCHAR` column to `TEXT` in the same invocation of `ndb_restore`.

Converting between `TEXT` columns using different character sets is not supported, and is expressly disallowed.

When performing conversions of character or binary types to `TEXT` or `BLOB` with `ndb_restore`, you may notice that it creates and uses one or more staging tables named *`table_name`*`$ST`*`node_id`*. These tables are not needed afterwards, and are normally deleted by `ndb_restore` following a successful restoration.

- `--rebuild-indexes`

| Command-Line Format | `--rebuild-indexes` |
|---|---|

Enable multithreaded rebuilding of the ordered indexes while restoring a native `NDB` backup. The number of threads used for building ordered indexes by `ndb_restore` with this option is controlled by the `BuildIndexThreads` data node configuration parameter and the number of LDMs.

It is necessary to use this option only for the first run of `ndb_restore`; this causes all ordered indexes to be rebuilt without using `--rebuild-indexes` again when restoring subsequent nodes. You should use this option prior to inserting new rows into the database; otherwise, it is possible for a row to be inserted that later causes a unique constraint violation when trying to rebuild the indexes.

Building of ordered indices is parallelized with the number of LDMs by default. Offline index builds performed during node and system restarts can be made faster using the `BuildIndexThreads` data node configuration parameter; this parameter has no effect on dropping and rebuilding of indexes by `ndb_restore`, which is performed online.

Rebuilding of unique indexes uses disk write bandwidth for redo logging and local checkpointing. An insufficient amount of this bandwidth can lead to redo buffer overload or log overload errors. In such cases you can run `ndb_restore --rebuild-indexes` again; the process resumes at the point where the error occurred. You can also do this when you have encountered temporary errors. You can repeat execution of `ndb_restore --rebuild-indexes` indefinitely; you may be able to stop such errors by reducing the value of `--parallelism`. If the problem is insufficient space, you can increase the size of the redo log (`FragmentLogFileSize` node configuration parameter), or you can increase the speed at which LCPs are performed (`MaxDiskWriteSpeed` and related parameters), in order to free space more quickly.

- `--remap-column=`*`db.tbl.col:fn:args`*

| Command-Line Format | `--remap-column=string` |
|---|---|
| Introduced | 8.0.21-ndb-8.0.21 |
| Type | String |

| Default Value | [none] |
| --- | --- |

When used together with --restore-data, this option applies a function to the value of the indicated column. Values in the argument string are listed here:

- *db*: Database name, following any renames performed by --rewrite-database.
- *tbl*: Table name.
- *col*: Name of the column to be updated. This column must be of type INT or BIGINT. The column can also be but is not required to be UNSIGNED.
- *fn*: Function name; currently, the only supported name is offset.
- *args*: Arguments supplied to the function. Currently, only a single argument, the size of the offset to be added by the offset function, is supported. Negative values are supported. The size of the argument cannot exceed that of the signed variant of the column's type; for example, if *col* is an INT column, then the allowed range of the argument passed to the offset function is -2147483648 to 2147483647 (see Section 11.1.2, “Integer Types (Exact Value) - INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT”).

  If applying the offset value to the column would cause an overflow or underflow, the restore operation fails. This could happen, for example, if the column is a BIGINT, and the option attempts to apply an offset value of 8 on a row in which the column value is 4294967291, since 4294967291 + 8 = 4294967299 > 4294967295.

This option can be useful when you wish to merge data stored in multiple source instances of NDB Cluster (all using the same schema) into a single destination NDB Cluster, using NDB native backup (see Section 23.6.8.2, “Using The NDB Cluster Management Client to Create a Backup”) and ndb_restore to merge the data, where primary and unique key values are overlapping between source clusters, and it is necessary as part of the process to remap these values to ranges that do not overlap. It may also be necessary to preserve other relationships between tables. To fulfill such requirements, it is possible to use the option multiple times in the same invocation of ndb_restore to remap columns of different tables, as shown here:

```
$> ndb_restore --restore-data --remap-column=hr.employee.id:offset:1000 \
    --remap-column=hr.manager.id:offset:1000 --remap-column=hr.firstaiders.id:offset:1000
```

(Other options not shown here may also be used.)

--remap-column can also be used to update multiple columns of the same table. Combinations of multiple tables and columns are possible. Different offset values can also be used for different columns of the same table, like this:

```
$> ndb_restore --restore-data --remap-column=hr.employee.salary:offset:10000 \
    --remap-column=hr.employee.hours:offset:-10
```

When source backups contain duplicate tables which should not be merged, you can handle this by using --exclude-tables, --exclude-databases, or by some other means in your application.

Information about the structure and other characteristics of tables to be merged can obtained using SHOW CREATE TABLE; the ndb_desc tool; and MAX(), MIN(), LAST_INSERT_ID(), and other MySQL functions.

Replication of changes from merged to unmerged tables, or from unmerged to merged tables, in separate instances of NDB Cluster is not supported.

- `--restore-data`, `-r`

| Command-Line Format | `--restore-data` |
|---|---|

Output `NDB` table data and logs.

- `--restore-epoch`, `-e`

| Command-Line Format | `--restore-epoch` |
|---|---|

Add (or restore) epoch information to the cluster replication status table. This is useful for starting replication on an NDB Cluster replica. When this option is used, the row in the `mysql.ndb_apply_status` having `0` in the `id` column is updated if it already exists; such a row is inserted if it does not already exist. (See Section 23.7.9, “NDB Cluster Backups With NDB Cluster Replication”.)

- `--restore-meta`, `-m`

| Command-Line Format | `--restore-meta` |
|---|---|

This option causes `ndb_restore` to print `NDB` table metadata.

The first time you run the `ndb_restore` restoration program, you also need to restore the metadata. In other words, you must re-create the database tables—this can be done by running it with the `--restore-meta` (`-m`) option. Restoring the metadata need be done only on a single data node; this is sufficient to restore it to the entire cluster.

In older versions of NDB Cluster, tables whose schemas were restored using this option used the same number of partitions as they did on the original cluster, even if it had a differing number of data nodes from the new cluster. In NDB 8.0, when restoring metadata, this is no longer an issue; `ndb_restore` now uses the default number of partitions for the target cluster, unless the number of local data manager threads is also changed from what it was for data nodes in the original cluster.

When using this option in NDB 8.0, it is recommended that auto synchronization be disabled by setting `ndb_metadata_check=OFF` until `ndb_restore` has completed restoring the metadata, after which it can it turned on again to synchronize objects newly created in the NDB dictionary.

**Note**

The cluster should have an empty database when starting to restore a backup. (In other words, you should start the data nodes with `--initial` prior to performing the restore.)

- `--restore-privilege-tables`

| Command-Line Format | `--restore-privilege-tables` |
|---|---|
| Deprecated | 8.0.16-ndb-8.0.16 |

`ndb_restore` does not by default restore distributed MySQL privilege tables created in releases of NDB Cluster prior to version 8.0, which does not support distributed privileges as implemented in NDB 7.6 and earlier. This option causes `ndb_restore` to restore them.

In NDB 8.0, such tables are not used for access control; as part of the MySQL server's upgrade process, the server creates `InnoDB` copies of these tables local to itself. For more information, see Section 23.3.7, “Upgrading and Downgrading NDB Cluster”, as well as Section 6.2.3, “Grant Tables”.



- `--rewrite-database=olddb,newdb`

| Command-Line Format | `--rewrite-database=string` |
|---|---|

| Type | String |
| --- | --- |
| Default Value | none |

This option makes it possible to restore to a database having a different name from that used in the backup. For example, if a backup is made of a database named products, you can restore the data it contains to a database named inventory, use this option as shown here (omitting any other options that might be required):

```
$> ndb_restore --rewrite-database=product,inventory
```

The option can be employed multiple times in a single invocation of ndb_restore. Thus it is possible to restore simultaneously from a database named db1 to a database named db2 and from a database named db3 to one named db4 using --rewrite-database=db1,db2 --rewrite-database=db3,db4. Other ndb_restore options may be used between multiple occurrences of --rewrite-database.

In the event of conflicts between multiple --rewrite-database options, the last --rewrite-database option used, reading from left to right, is the one that takes effect. For example, if --rewrite-database=db1,db2 --rewrite-database=db1,db3 is used, only --rewrite-database=db1,db3 is honored, and --rewrite-database=db1,db2 is ignored. It is also possible to restore from multiple databases to a single database, so that --rewrite-database=db1,db3 --rewrite-database=db2,db3 restores all tables and data from databases db1 and db2 into database db3.

**Important**

When restoring from multiple backup databases into a single target database using --rewrite-database, no check is made for collisions between table or other object names, and the order in which rows are restored is not guaranteed. This means that it is possible in such cases for rows to be overwritten and updates to be lost.

- --skip-broken-objects

| Command-Line Format | --skip-broken-objects |
| --- | --- |

This option causes ndb_restore to ignore corrupt tables while reading a native NDB backup, and to continue restoring any remaining tables (that are not also corrupted). Currently, the --skip-broken-objects option works only in the case of missing blob parts tables.

- --skip-table-check, -s

| Command-Line Format | --skip-table-check |
| --- | --- |

It is possible to restore data without restoring table metadata. By default when doing this, ndb_restore fails with an error if a mismatch is found between the table data and the table schema; this option overrides that behavior.

Some of the restrictions on mismatches in column definitions when restoring data using ndb_restore are relaxed; when one of these types of mismatches is encountered, ndb_restore does not stop with an error as it did previously, but rather accepts the data and inserts it into the target table while issuing a warning to the user that this is being done. This behavior occurs whether

or not either of the options `--skip-table-check` or `--promote-attributes` is in use. These differences in column definitions are of the following types:

- Different `COLUMN_FORMAT` settings (`FIXED`, `DYNAMIC`, `DEFAULT`)
- Different `STORAGE` settings (`MEMORY`, `DISK`)
- Different default values
- Different distribution key settings

- `--skip-unknown-objects`

| Command-Line Format | `--skip-unknown-objects` |
|---|---|

This option causes `ndb_restore` to ignore any schema objects it does not recognize while reading a native `NDB` backup. This can be used for restoring a backup made from a cluster running (for example) NDB 7.6 to a cluster running NDB Cluster 7.5.

- `--slice-id=#`

| Command-Line Format | `--slice-id=#` |
|---|---|
| Introduced | 8.0.20-ndb-8.0.20 |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `1023` |

When restoring by slices, this is the ID of the slice to restore. This option is always used together with `--num-slices`, and its value must be always less than that of `--num-slices`.

For more information, see the description of the `--num-slices` elsewhere in this section.

- `--tab=`*`dir_name`*, `-T` *`dir_name`*

| Command-Line Format | `--tab=path` |
|---|---|
| Type | Directory name |

Causes `--print-data` to create dump files, one per table, each named *`tbl_name`*`.txt`. It requires as its argument the path to the directory where the files should be saved; use `.` for the current directory.

- `--timestamp-printouts`

| Command-Line Format | `--timestamp-printouts{=true\|false}` |
|---|---|
| Introduced | 8.0.33-ndb-8.0.33 |
| Type | Boolean |
| Default Value | `true` |

Causes info, error, and debug log messages to be prefixed with timestamps.

This option is enabled by default in NDB 8.0. Disable it with `--timestamp-printouts=false`.

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

Display help text and exit; same as `--help`.

- `--verbose=#`

| Command-Line Format | `--verbose=#` |
|---|---|
| Type | Numeric |
| Default Value | `1` |
| Minimum Value | `0` |
| Maximum Value | `255` |

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

- `--with-apply-status`

| Command-Line Format | `--with-apply-status` |
|---|---|
| Introduced | 8.0.29-ndb-8.0.29 |

Restore all rows from the backup's `ndb_apply_status` table (except for the row having `server_id = 0`, which is generated using `--restore-epoch`). This option requires that `--restore-data` also be used.

If the `ndb_apply_status` table from the backup already contains a row with `server_id = 0`, `ndb_restore --with-apply-status` deletes it. For this reason, we recommend that you use `ndb_restore --restore-epoch` after invoking `ndb_restore` with the `--with-apply-status` option. You can also use `--restore-epoch` concurrently with the last of any invocations of `ndb_restore --with-apply-status` used to restore the cluster.

For more information, see ndb_apply_status Table.

Typical options for this utility are shown here:

```
ndb_restore [-c connection_string] -n node_id -b backup_id \
      [-m] -r --backup-path=/path/to/backup/files
```

Normally, when restoring from an NDB Cluster backup, `ndb_restore` requires at a minimum the `--nodeid` (short form: `-n`), `--backupid` (short form: `-b`), and `--backup-path` options.

The `-c` option is used to specify a connection string which tells `ndb_restore` where to locate the cluster management server (see Section 23.4.3.3, “NDB Cluster Connection Strings”). If this option is not used, then `ndb_restore` attempts to connect to a management server on `localhost:1186`. This utility acts as a cluster API node, and so requires a free connection “slot” to connect to the cluster management server. This means that there must be at least one `[api]` or `[mysqld]` section that can be used by it in the cluster `config.ini` file. It is a good idea to keep at least one empty `[api]` or `[mysqld]` section in `config.ini` that is not being used for a MySQL server or other application for this reason (see Section 23.4.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”).

In NDB 8.0.22 and later, `ndb_restore` can decrypt an encrypted backup using `--decrypt` and `--backup-password`. Both options must be specified to perform decryption. See the documentation for the `START BACKUP` management client command for information on creating encrypted backups.

You can verify that `ndb_restore` is connected to the cluster by using the `SHOW` command in the `ndb_mgm` management client. You can also accomplish this from a system shell, as shown here:

```
$> ndb_mgm -e "SHOW"
```

**Error reporting.**
`ndb_restore` reports both temporary and permanent errors. In the case of temporary errors, it may able to recover from them, and reports `Restore successful, but encountered temporary error, please look at configuration` in such cases.

**Important**

After using `ndb_restore` to initialize an NDB Cluster for use in circular replication, binary logs on the SQL node acting as the replica are not automatically created, and you must cause them to be created manually. To cause the binary logs to be created, issue a `SHOW TABLES` statement on that SQL node before running `START SLAVE`. This is a known issue in NDB Cluster.

### 23.5.23.1 Restoring an NDB Backup to a Different Version of NDB Cluster

The following two sections provide information about restoring a native NDB backup to a different version of NDB Cluster from the version in which the backup was taken.

In addition, you should consult Section 23.3.7, “Upgrading and Downgrading NDB Cluster”, for other issues you may encounter when attempting to restore an NDB backup to a cluster running a different version of the NDB software.

It is also advisable to review What is New in NDB Cluster 8.0, as well as Section 2.10.4, “Changes in MySQL 8.0”, for other changes between NDB 8.0 and previous versions of NDB Cluster that may be relevant to your particular circumstances.

#### Restoring an NDB backup to a previous version of NDB Cluster

You may encounter issues when restoring a backup taken from a later version of NDB Cluster to a previous one, due to the use of features which do not exist in the earlier version. Some of these issues are listed here:

- **utf8mb4_ai_ci character set.** Tables created in NDB 8.0 by default use the `utf8mb4_ai_ci` character set, which is not available in NDB 7.6 and earlier, and so cannot be read by an `ndb_restore` binary from one of these earlier versions. In such cases, it is necessary to alter any tables using `utf8mb4_ai_ci` so that they use a character set supported in the older version prior to performing the backup.

- **Table metadata format.** Due to changes in how the MySQL Server and NDB handle table metadata, tables created or altered using the included MySQL server binary from NDB 8.0 cannot be restored using `ndb_restore` to NDB 7.6 or an earlier version of NDB Cluster. Such tables use `.sdi` files which are not understood by older versions of `mysqld`.

  A backup taken in NDB 8.0 of tables which were created in NDB 7.6 or earlier, and which have not been altered since upgrading to NDB 8.0, should be restorable to older versions of NDB Cluster.

  Since it is possible to restore metadata and table data separately, you can in such cases restore the table schemas from a dump made using `mysqldump`, or by executing the necessary `CREATE TABLE` statements manually, then import only the table data using `ndb_restore` with the `--restore-data` option.

- **Multi-threaded backups.** Multi-threaded backups taken in NDB 8.0 can be restored to an cluster running an earlier version of `NDB` in either of the following two ways:

  - Using an `ndb_restore` binary from NDB 8.0, perform a parallel restore. See Restoring a parallel backup in parallel.
  - Restore the backups serially; in this case, a later version of `ndb_restore` is not required. See Restoring a parallel backup serially.
- **Encrypted backups.** Encrypted backups created in NDB 8.0.22 and later cannot be restored using `ndb_restore` from NDB 8.0.21 or earlier.
- **NDB_STORED_USER privilege.** The `NDB_STORED_USER` privilege is supported only in NDB 8.0.
- **Maximum number of data nodes.** NDB Cluster 8.0 supports up to 144 data nodes, while earlier versions support a maximum of only 48 data nodes. See Restoring to Fewer Nodes Than the Original, for information with situations in which this incompatibility causes an issue.

#### Restoring an NDB backup to a later version of NDB Cluster

In general, it should be possible to restore a backup created using the `ndb_mgm` client `START BACKUP` command in an older version of NDB to a newer version, provided that you use the `ndb_restore` binary that comes with the newer version. (It may be possible to use the older version of `ndb_restore`, but this is not recommended.) Additional potential issues are listed here:

- When restoring the metadata from a backup (`--restore-meta` option), `ndb_restore` normally attempts to reproduce the captured table schema exactly as it was when the backup was taken.

  Tables created in versions of NDB prior to 8.0 use `.frm` files for their metadata. These files can be read by the `mysqld` in NDB 8.0, which can use the information contained therein to create the `.sdi` files used by the MySQL data dictionary in later versions.
- When restoring an older backup to a newer version of NDB, it may not be possible to take advantage of newer features such as hashmap partitioning, greater number of hashmap buckets, read backup, and different partitioning layouts. For this reason, it may be preferable to restore older schemas using `mysqldump` and the `mysql` client, which allows NDB to make use of the new schema features.
- Tables using the old temporal types which did not support fractional seconds (used prior to MySQL 5.6.4 and NDB 7.3.31) cannot be restored to NDB 8.0 using `ndb_restore`. You can check such tables using `CHECK TABLE`, and then upgrade them to the newer temporal column format, if necessary, using `REPAIR TABLE` in the `mysql` client; this must be done prior to taking the backup. See Section 2.10.5, “Preparing Your Installation for Upgrade”, for more information.

  You also restore such tables using a dump created with `mysqldump`.
- Distributed grant tables created in NDB 7.6 and earlier are not supported in NDB 8.0. Such tables can be restored to an NDB 8.0 cluster, but they have no effect on access control.

### 23.5.23.2 Restoring to a different number of data nodes

It is possible to restore from an NDB backup to a cluster having a different number of data nodes than the original from which the backup was taken. The following two sections discuss, respectively, the cases where the target cluster has a lesser or greater number of data nodes than the source of the backup.

#### Restoring to Fewer Nodes Than the Original

You can restore to a cluster having fewer data nodes than the original provided that the larger number of nodes is an even multiple of the smaller number. In the following example, we use a backup taken on a cluster having four data nodes to a cluster having two data nodes.

1. The management server for the original cluster is on host `host10`. The original cluster has four data nodes, with the node IDs and host names shown in the following extract from the management server's `config.ini` file:

```
[ndbd]
NodeId=2
HostName=host2

[ndbd]
NodeId=4
HostName=host4

[ndbd]
NodeId=6
HostName=host6

[ndbd]
NodeId=8
HostName=host8
```

We assume that each data node was originally started with `ndbmtd --ndb-connectstring=host10` or the equivalent.

2. Perform a backup in the normal manner. See Section 23.6.8.2, “Using The NDB Cluster Management Client to Create a Backup”, for information about how to do this.

3. The files created by the backup on each data node are listed here, where *`N`* is the node ID and *`B`* is the backup ID.

   - `BACKUP-B-0.N.Data`
   - `BACKUP-B.N.ctl`
   - `BACKUP-B.N.log`

   These files are found under `BackupDataDir/BACKUP/BACKUP-B`, on each data node. For the rest of this example, we assume that the backup ID is 1.

   Have all of these files available for later copying to the new data nodes (where they can be accessed on the data node's local file system by `ndb_restore`). It is simplest to copy them all to a single location; we assume that this is what you have done.

4. The management server for the target cluster is on host `host20`, and the target has two data nodes, with the node IDs and host names shown, from the management server `config.ini` file on `host20`:

   ```
   [ndbd]
   NodeId=3
   hostname=host3

   [ndbd]
   NodeId=5
   hostname=host5
   ```

   Each of the data node processes on `host3` and `host5` should be started with `ndbmtd -c host20 --initial` or the equivalent, so that the new (target) cluster starts with clean data node file systems.

5. Copy two different sets of two backup files to each of the target data nodes. For this example, copy the backup files from nodes 2 and 4 from the original cluster to node 3 in the target cluster. These files are listed here:

   - `BACKUP-1-0.2.Data`
   - `BACKUP-1.2.ctl`
   - `BACKUP-1.2.log`
   - `BACKUP-1-0.4.Data`

- BACKUP-1.4.ctl
- BACKUP-1.4.log

Then copy the backup files from nodes 6 and 8 to node 5; these files are shown in the following list:

- BACKUP-1-0.6.Data
- BACKUP-1.6.ctl
- BACKUP-1.6.log
- BACKUP-1-0.8.Data
- BACKUP-1.8.ctl
- BACKUP-1.8.log

For the remainder of this example, we assume that the respective backup files have been saved to the directory /BACKUP-1 on each of nodes 3 and 5.

6. On each of the two target data nodes, you must restore from both sets of backups. First, restore the backups from nodes 2 and 4 to node 3 by invoking ndb_restore on host3 as shown here:

```
$> ndb_restore -c host20 --nodeid=2 --backupid=1 --restore-data --backup-path=/BACKUP-1

$> ndb_restore -c host20 --nodeid=4 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

Then restore the backups from nodes 6 and 8 to node 5 by invoking ndb_restore on host5, like this:

```
$> ndb_restore -c host20 --nodeid=6 --backupid=1 --restore-data --backup-path=/BACKUP-1

$> ndb_restore -c host20 --nodeid=8 --backupid=1 --restore-data --backup-path=/BACKUP-1
```

### Restoring to More Nodes Than the Original

The node ID specified for a given ndb_restore command is that of the node in the original backup and not that of the data node to restore it to. When performing a backup using the method described in this section, ndb_restore connects to the management server and obtains a list of data nodes in the cluster the backup is being restored to. The restored data is distributed accordingly, so that the number of nodes in the target cluster does not need to be to be known or calculated when performing the backup.

**Note**

When changing the total number of LCP threads or LQH threads per node group, you should recreate the schema from backup created using mysqldump.

1. *Create the backup of the data*. You can do this by invoking the ndb_mgm client START BACKUP command from the system shell, like this:

```
$> ndb_mgm -e "START BACKUP 1"
```

This assumes that the desired backup ID is 1.

2. Create a backup of the schema. This step is necessary only if the total number of LCP threads or LQH threads per node group is changed.

```
$> mysqldump --no-data --routines --events --triggers --databases > myschema.sql
```

**Important**

Once you have created the NDB native backup using ndb_mgm, you must not make any schema changes before creating the backup of the schema, if you do so.

3. Copy the backup directory to the new cluster. For example if the backup you want to restore has ID 1 and BackupDataDir = /backups/node_*nodeid*, then the path to the backup on this node is /backups/node_1/BACKUP/BACKUP-1. Inside this directory there are three files, listed here:

   - BACKUP-1-0.1.Data
   - BACKUP-1.1.ctl
   - BACKUP-1.1.log

   You should copy the entire directory to the new node.

   If you needed to create a schema file, copy this to a location on an SQL node where it can be read by mysqld.

There is no requirement for the backup to be restored from a specific node or nodes.

To restore from the backup just created, perform the following steps:

1. *Restore the schema*.

   - If you created a separate schema backup file using mysqldump, import this file using the mysql client, similar to what is shown here:

     ```
     $> mysql < myschema.sql
     ```

     When importing the schema file, you may need to specify the --user and --password options (and possibly others) in addition to what is shown, in order for the mysql client to be able to connect to the MySQL server.

   - If you did *not* need to create a schema file, you can re-create the schema using ndb_restore --restore-meta (short form -m), similar to what is shown here:

     ```
     $> ndb_restore --nodeid=1 --backupid=1 --restore-meta --backup-path=/backups/node_1/BACKUP/BACKUP-1
     ```

     ndb_restore must be able to contact the management server; add the --ndb-connectstring option if and as needed to make this possible.

2. *Restore the data*. This needs to be done once for each data node in the original cluster, each time using that data node's node ID. Assuming that there were 4 data nodes originally, the set of commands required would look something like this:

   ```
   ndb_restore --nodeid=1 --backupid=1 --restore-data --backup-path=/backups/node_1/BACKUP/BACKUP-1 --disal
   ndb_restore --nodeid=2 --backupid=1 --restore-data --backup-path=/backups/node_2/BACKUP/BACKUP-1 --disal
   ndb_restore --nodeid=3 --backupid=1 --restore-data --backup-path=/backups/node_3/BACKUP/BACKUP-1 --disal
   ndb_restore --nodeid=4 --backupid=1 --restore-data --backup-path=/backups/node_4/BACKUP/BACKUP-1 --disal
   ```

   These can be run in parallel.

   Be sure to add the --ndb-connectstring option as needed.

3. *Rebuild the indexes*. These were disabled by the --disable-indexes option used in the commands just shown. Recreating the indexes avoids errors due to the restore not being consistent at all points. Rebuilding the indexes can also improve performance in some cases. To rebuild the indexes, execute the following command once, on a single node:

   ```
   $> ndb_restore --nodeid=1 --backupid=1 --backup-path=/backups/node_1/BACKUP/BACKUP-1 --rebuild-indexes
   ```

As mentioned previously, you may need to add the `--ndb-connectstring` option, so that `ndb_restore` can contact the management server.

### 23.5.23.3 Restoring from a backup taken in parallel

NDB Cluster 8.0 supports parallel backups on each data node using `ndbmtd` with multiple LDMs (see Section 23.6.8.5, "Taking an NDB Backup with Parallel Data Nodes"). The next two sections describe how to restore backups that were taken in this fashion.

#### Restoring a parallel backup in parallel

Restoring a parallel backup in parallel requires an `ndb_restore` binary from an NDB 8.0 distribution. The process is not substantially different from that outlined in the general usage section under the description of the `ndb_restore` program, and consists of executing `ndb_restore` twice, similarly to what is shown here:

```
$> ndb_restore -n 1 -b 1 -m --backup-path=path/to/backup_dir/BACKUP/BACKUP-backup_id
$> ndb_restore -n 1 -b 1 -r --backup-path=path/to/backup_dir/BACKUP/BACKUP-backup_id
```

*`backup_id`* is the ID of the backup to be restored. In the general case, no additional special arguments are required; `ndb_restore` always checks for the existence of parallel subdirectories under the directory indicated by the `--backup-path` option and restores the metadata (serially) and then the table data (in parallel).

#### Restoring a parallel backup serially

It is possible to restore a backup that was made using parallelism on the data nodes in serial fashion. To do this, invoke `ndb_restore` with `--backup-path` pointing to the subdirectories created by each LDM under the main backup directory, once to any one of the subdirectories to restore the metadata (it does not matter which one, since each subdirectory contains a complete copy of the metadata), then to each of the subdirectories in turn to restore the data. Suppose that we want to restore the backup having backup ID 100 that was taken with four LDMs, and that the `BackupDataDir` is `/opt`. To restore the metadata in this case, we can invoke `ndb_restore` like this:

```
$> ndb_restore -n 1 -b 1 -m --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
```

To restore the table data, execute `ndb_restore` four times, each time using one of the subdirectories in turn, as shown here:

```
$> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
$> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-2-OF-4
$> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-3-OF-4
$> ndb_restore -n 1 -b 1 -r --backup-path=opt/BACKUP/BACKUP-100/BACKUP-100-PART-4-OF-4
```

You can employ the same technique to restore a parallel backup to an older version of NDB Cluster (7.6 or earlier) that does not support parallel backups, using the `ndb_restore` binary supplied with the older version of the NDB Cluster software.

## 23.5.24 ndb_secretsfile_reader — Obtain Key Information from an Encrypted NDB Data File

`ndb_secretsfile_reader` gets the encryption key from an `NDB` encryption secrets file, given the password.

### Usage

```
ndb_secretsfile_reader options file
```

The *`options`* must include one of `--filesystem-password` or `--filesystem-password-from-stdin`, and the encryption password must be supplied, as shown here:

```
> ndb_secretsfile_reader --filesystem-password=54kl14 ndb_5_fs/D1/NDBCNTR/S0.sysfile
```

```
ndb_secretsfile_reader: [Warning] Using a password on the command line interface can be insecure.
cac256e18b2ddf6b5ef82d99a72f18e864b78453cc7fa40bfaf0c40b91122d18
```

These and other options that can be used with `ndb_secretsfile_reader` are shown in the following table. Additional descriptions follow the table.

**Table 23.45 Command-line options used with the program ndb_secretsfile_reader**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--filesystem-password=password` | Password for node file system encryption; can be passed from stdin, tty, or my.cnf file | ADDED: 8.0.31 |
| `--filesystem-password-from-stdin={TRUE\|FALSE}` | Get encryption password from stdin | ADDED: 8.0.31 |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--filesystem-password`

| Command-Line Format | `--filesystem-password=password` |
| --- | --- |
| Introduced | 8.0.31 |

Pass the filesystem encryption and decryption password to `ndb_secretsfile_reader` using `stdin`, `tty`, or the `my.cnf` file.

- `--filesystem-password-from-stdin`

| Command-Line Format | `--filesystem-password-from-stdin={TRUE\|FALSE}` |
| --- | --- |
| Introduced | 8.0.31 |

Pass the filesystem encryption and decryption password to `ndb_secretsfile_reader` from `stdin` (only).

- `--help`

| Command-Line Format | `--help` |
| --- | --- |

Display help text and exit.

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as --help.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

`ndb_secretsfile_reader` was added in NDB 8.0.31.

## 23.5.25 ndb_select_all — Print Rows from an NDB Table

`ndb_select_all` prints all rows from an `NDB` table to `stdout`.

### Usage

```
ndb_select_all -c connection_string tbl_name -d db_name [> file_name]
```

Options that can be used with `ndb_select_all` are shown in the following table. Additional descriptions follow the table.

**Table 23.46 Command-line options used with the program ndb_select_all**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database in which table is found | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--delimiter=char`, `-D char` | Set column delimiter | (Supported in all NDB releases based on MySQL 8.0) |
| `--descending`, `-z` | Sort resultset in descending order (requires --order) | (Supported in all NDB releases based on MySQL 8.0) |
| `--disk` | Print disk references (useful only for Disk Data tables having unindexed columns) | (Supported in all NDB releases based on MySQL 8.0) |
| `--gci` | Include GCI in output | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--gci64` | Include GCI and row epoch in output | (Supported in all NDB releases based on MySQL 8.0) |
| `--header[=value]`, `-h` | Print header (set to 0\|FALSE to disable headers in output) | (Supported in all NDB releases based on MySQL 8.0) |
| `--lock=#`, `-l #` | Lock type | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--nodata` | Do not print table column data | (Supported in all NDB releases based on MySQL 8.0) |
| `--order=index`, `-o index` | Sort resultset according to index having this name | (Supported in all NDB releases based on MySQL 8.0) |
| `--parallelism=#`, `-p #` | Degree of parallelism | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--rowid` | Print row ID | (Supported in all NDB releases based on MySQL 8.0) |
| `--tupscan`, `-t` | Scan in tup order | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
| --- | --- | --- |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--useHexFormat`, `-x` | Output numbers in hexadecimal format | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
| --- | --- |
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
| --- | --- |
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
| --- | --- |
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
| --- | --- |
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--database=dbname`, `-d dbname`

  Name of the database in which the table is found. The default value is `TEST_DB`.

- `--descending`, `-z`

  Sorts the output in descending order. This option can be used only in conjunction with the `-o` (`--order`) option.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Also read groups with concat(group, suffix).

- `--delimiter=character`, `-D character`

  Causes the *`character`* to be used as a column delimiter. Only table data columns are separated by this delimiter.

  The default delimiter is the tab character.

- `--disk`

  Adds a disk reference column to the output. The column is nonempty only for Disk Data tables having nonindexed columns.

- `--gci`

  Adds a `GCI` column to the output showing the global checkpoint at which each row was last updated. See Section 23.2, “NDB Cluster Overview”, and Section 23.6.3.2, “NDB Cluster Log Events”, for more information about checkpoints.

- `--gci64`

  Adds a `ROW$GCI64` column to the output showing the global checkpoint at which each row was last updated, as well as the number of the epoch in which this update occurred.

- `--help`

| Command-Line Format | `--help` |
| --- | --- |

Display help text and exit.

- `--lock=lock_type`, `-l lock_type`

  Employs a lock when reading the table. Possible values for `lock_type` are:

  - `0`: Read lock
  - `1`: Read lock with hold
  - `2`: Exclusive read lock

  There is no default value for this option.

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Read given path from login file.

- `--header=FALSE`

  Excludes column headers from the output.

- `--nodata`

  Causes any table data to be omitted.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

  Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

  Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--order=index_name`, `-o index_name`

Orders the output according to the index named *`index_name`*.

**Note**

This is the name of an index, not of a column; the index must have been explicitly named when created.

- `parallelism=#`, `-p #`

Specifies the degree of parallelism.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--rowid`

Adds a `ROWID` column providing information about the fragments in which rows are stored.

- `--tupscan`, `-t`

Scan the table in the order of the tuples.

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as `--help`.

- `--useHexFormat -x`

Causes all numeric values to be displayed in hexadecimal format. This does not affect the output of numerals contained in strings or datetime values.

- `--version`

| Command-Line Format | `--version` |
| --- | --- |

Display version information and exit.

## Sample Output

Output from a MySQL `SELECT` statement:

```
mysql> SELECT * FROM ctest1.fish;
```

```
+----+-----------+
| id | name      |
+----+-----------+
|  3 | shark     |
|  6 | puffer    |
|  2 | tuna      |
|  4 | manta ray |
|  5 | grouper   |
|  1 | guppy     |
+----+-----------+
6 rows in set (0.04 sec)
```

Output from the equivalent invocation of `ndb_select_all`:

```
$> ./ndb_select_all -c localhost fish -d ctest1
id      name
3       [shark]
6       [puffer]
2       [tuna]
4       [manta ray]
5       [grouper]
1       [guppy]
6 rows returned

NDBT_ProgramExit: 0 - OK
```

All string values are enclosed by square brackets (`[...]`) in the output of `ndb_select_all`. For another example, consider the table created and populated as shown here:

```
CREATE TABLE dogs (
    id INT(11) NOT NULL AUTO_INCREMENT,
    name VARCHAR(25) NOT NULL,
    breed VARCHAR(50) NOT NULL,
    PRIMARY KEY pk (id),
    KEY ix (name)
)
TABLESPACE ts STORAGE DISK
ENGINE=NDBCLUSTER;

INSERT INTO dogs VALUES
    ('', 'Lassie', 'collie'),
    ('', 'Scooby-Doo', 'Great Dane'),
    ('', 'Rin-Tin-Tin', 'Alsatian'),
    ('', 'Rosscoe', 'Mutt');
```

This demonstrates the use of several additional `ndb_select_all` options:

```
$> ./ndb_select_all -d ctest1 dogs -o ix -z --gci --disk
GCI     id name          breed        DISK_REF
834461  2  [Scooby-Doo]  [Great Dane] [ m_file_no: 0 m_page: 98 m_page_idx: 0 ]
834878  4  [Rosscoe]     [Mutt]       [ m_file_no: 0 m_page: 98 m_page_idx: 16 ]
834463  3  [Rin-Tin-Tin] [Alsatian]   [ m_file_no: 0 m_page: 34 m_page_idx: 0 ]
835657  1  [Lassie]      [Collie]     [ m_file_no: 0 m_page: 66 m_page_idx: 0 ]
4 rows returned

NDBT_ProgramExit: 0 - OK
```

## 23.5.26 ndb_select_count — Print Row Counts for NDB Tables

`ndb_select_count` prints the number of rows in one or more `NDB` tables. With a single table, the result is equivalent to that obtained by using the MySQL statement `SELECT COUNT(*) FROM tbl_name`.

### Usage

```
ndb_select_count [-c connection_string] -ddb_name tbl_name[, tbl_name2[, ...]]
```

Options that can be used with `ndb_select_count` are shown in the following table. Additional descriptions follow the table.

**Table 23.47 Command-line options used with the program ndb_select_count**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Name of database in which table is found | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--lock=#`, `-l #` | Lock type | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--parallelism=#`, `-p #` | Degree of parallelism | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
|---|---|
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
|---|---|
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
| --- | --- |
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--help`

| Command-Line Format | `--help` |
| --- | --- |

Display help text and exit.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--usage`

| Command-Line Format | `--usage` |
| --- | --- |

Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
| --- | --- |

Display version information and exit.

You can obtain row counts from multiple tables in the same database by listing the table names separated by spaces when invoking this command, as shown under **Sample Output**.

### Sample Output

```
$> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs

NDBT_ProgramExit: 0 - OK
```

## 23.5.27 ndb_show_tables — Display List of NDB Tables

`ndb_show_tables` displays a list of all `NDB` database objects in the cluster. By default, this includes not only both user-created tables and `NDB` system tables, but `NDB`-specific indexes, internal triggers, and NDB Cluster Disk Data objects as well.

Options that can be used with `ndb_show_tables` are shown in the following table. Additional descriptions follow the table.

**Table 23.48 Command-line options used with the program ndb_show_tables**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--database=name`, `-d name` | Specifies database in which table is found; database name must be followed by table name | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--loops=#`, `-l #` | Number of times to repeat output | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--parsable`, `-p` | Return output suitable for MySQL LOAD DATA statement | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--show-temp-status` | Show table temporary flag | (Supported in all NDB releases based on MySQL 8.0) |
| `--type=#`, `-t #` | Limit output to objects of this type | (Supported in all NDB releases based on MySQL 8.0) |
| `--unqualified`, `-u` | Do not qualify table names | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
ndb_show_tables [-c connection_string]
```

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
|---|---|
| Type | Integer |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
|---|---|

| Type | Integer |
| --- | --- |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
| --- | --- |
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--database`, `-d`

Specifies the name of the database in which the desired table is found. If this option is given, the name of a table must follow the database name.

If this option has not been specified, and no tables are found in the `TEST_DB` database, `ndb_show_tables` issues a warning.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--help`

| Command-Line Format | `--help` |
|---|---|

Display help text and exit.

- `--login-path`

| Command-Line Format | `--login-path=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--loops`, `-l`

Specifies the number of times the utility should execute. This is 1 when this option is not specified, but if you do use the option, you must supply an integer argument for it.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
|---|---|
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
|---|---|
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
|---|---|
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
|---|---|

  Do not read default options from any option file other than login file.

- `--parsable`, `-p`

  Using this option causes the output to be in a format suitable for use with `LOAD DATA`.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

  Print program argument list and exit.

- `--show-temp-status`

  If specified, this causes temporary tables to be displayed.

- `--type`, `-t`

  Can be used to restrict the output to one type of object, specified by an integer type code as shown here:

  - `1`: System table
  - `2`: User-created table
  - `3`: Unique hash index

  Any other value causes all `NDB` database objects to be listed (the default).

- `--unqualified`, `-u`

  If specified, this causes unqualified object names to be displayed.

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

  Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
|---|---|

  Display version information and exit.

**Note**

Only user-created NDB Cluster tables may be accessed from MySQL; system tables such as `SYSTAB_0` are not visible to `mysqld`. However, you can examine the contents of system tables using `NDB` API applications such as `ndb_select_all` (see Section 23.5.25, “ndb_select_all — Print Rows from an NDB Table”).

Prior to NDB 8.0.20, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency has been removed, eliminating the extraneous output.

## 23.5.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator

This is a Perl script that can be used to estimate the amount of space that would be required by a MySQL database if it were converted to use the `NDBCLUSTER` storage engine. Unlike the other utilities discussed in this section, it does not require access to an NDB Cluster (in fact, there is no reason for it to do so). However, it does need to access the MySQL server on which the database to be tested resides.

## Requirements

- A running MySQL server. The server instance does not have to provide support for NDB Cluster.
- A working installation of Perl.
- The `DBI` module, which can be obtained from CPAN if it is not already part of your Perl installation. (Many Linux and other operating system distributions provide their own packages for this library.)
- A MySQL user account having the necessary privileges. If you do not wish to use an existing account, then creating one using `GRANT USAGE ON db_name.*`—where *`db_name`* is the name of the database to be examined—is sufficient for this purpose.

`ndb_size.pl` can also be found in the MySQL sources in `storage/ndb/tools`.

Options that can be used with `ndb_size.pl` are shown in the following table. Additional descriptions follow the table.

**Table 23.49 Command-line options used with the program ndb_size.pl**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--database=string` | Database or databases to examine; a comma-delimited list; default is ALL (use all databases found on server) | (Supported in all NDB releases based on MySQL 8.0) |
| `--hostname=string` | Specify host and optional port in host[:port] format | (Supported in all NDB releases based on MySQL 8.0) |
| `--socket=path` | Specify socket to connect to | (Supported in all NDB releases based on MySQL 8.0) |
| `--user=string` | Specify MySQL user name | (Supported in all NDB releases based on MySQL 8.0) |
| `--password=password` | Specify MySQL user password | (Supported in all NDB releases based on MySQL 8.0) |
| `--format=string` | Set output format (text or HTML) | (Supported in all NDB releases based on MySQL 8.0) |
| `--excludetables=list` | Skip any tables in comma-separated list | (Supported in all NDB releases based on MySQL 8.0) |
| `--excludedbs=list` | Skip any databases in comma-separated list | (Supported in all NDB releases based on MySQL 8.0) |
| `--savequeries=path` | Saves all queries on database into file specified | (Supported in all NDB releases based on MySQL 8.0) |
| `--loadqueries=path` | Loads all queries from file specified; does not connect to database | (Supported in all NDB releases based on MySQL 8.0) |
| `--real_table_name=string` | Designates table to handle unique index size calculations | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
perl ndb_size.pl [--database={db_name|ALL}] [--hostname=host[:port]] [--socket=socket] \
```

```
      [--user=user] [--password=password]  \
      [--help|-h] [--format={html|text}] \
      [--loadqueries=file_name] [--savequeries=file_name]
```

By default, this utility attempts to analyze all databases on the server. You can specify a single database using the `--database` option; the default behavior can be made explicit by using `ALL` for the name of the database. You can also exclude one or more databases by using the `--excludedbs` option with a comma-separated list of the names of the databases to be skipped. Similarly, you can cause specific tables to be skipped by listing their names, separated by commas, following the optional `--excludetables` option. A host name can be specified using `--hostname`; the default is `localhost`. You can specify a port in addition to the host using *`host`*:*`port`* format for the value of `--hostname`. The default port number is 3306. If necessary, you can also specify a socket; the default is `/var/lib/mysql.sock`. A MySQL user name and password can be specified the corresponding options shown. It also possible to control the format of the output using the `--format` option; this can take either of the values `html` or `text`, with `text` being the default. An example of the text output is shown here:

```
$> ndb_size.pl --database=test --socket=/tmp/mysql.sock
ndb_size.pl report for database: 'test' (1 tables)
----------------------------------------------------
Connected to: DBI:mysql:host=localhost;mysql_socket=/tmp/mysql.sock

Including information for versions: 4.1, 5.0, 5.1

test.t1
-------

DataMemory for Columns (* means varsized DataMemory):
         Column Name            Type  Varsized   Key  4.1  5.0   5.1
     HIDDEN_NDB_PKEY          bigint             PRI    8    8     8
                  c2     varchar(50)         Y         52   52    4*
                  c1         int(11)                    4    4     4
                                                       --   --    --
Fixed Size Columns DM/Row                              64   64    12
   Varsize Columns DM/Row                               0    0     4

DataMemory for Indexes:
   Index Name                Type        4.1        5.0        5.1
      PRIMARY               BTREE         16         16         16
                                          --         --         --
       Total Index DM/Row                 16         16         16

IndexMemory for Indexes:
              Index Name        4.1        5.0        5.1
                 PRIMARY         33         16         16
                                 --         --         --
          Indexes IM/Row         33         16         16

Summary (for THIS table):
                                4.1        5.0        5.1
    Fixed Overhead DM/Row        12         12         16
           NULL Bytes/Row         4          4          4
           DataMemory/Row        96         96         48
                    (Includes overhead, bitmap and indexes)

  Varsize Overhead DM/Row          0          0          8
   Varsize NULL Bytes/Row          0          0          4
       Avg Varside DM/Row          0          0         16

                 No. Rows          0          0          0

        Rows/32kb DM Page        340        340        680
Fixedsize DataMemory (KB)          0          0          0

Rows/32kb Varsize DM Page          0          0       2040
  Varsize DataMemory (KB)          0          0          0

         Rows/8kb IM Page        248        512        512
         IndexMemory (KB)          0          0          0
```

```
Parameter Minimum Requirements
------------------------------
* indicates greater than default

               Parameter     Default        4.1         5.0         5.1
         DataMemory (KB)       81920          0           0           0
      NoOfOrderedIndexes         128          1           1           1
              NoOfTables         128          1           1           1
         IndexMemory (KB)       18432          0           0           0
   NoOfUniqueHashIndexes          64          0           0           0
           NoOfAttributes        1000          3           3           3
             NoOfTriggers         768          5           5           5
```

For debugging purposes, the Perl arrays containing the queries run by this script can be read from the file specified using can be saved to a file using `--savequeries`; a file containing such arrays to be read during script execution can be specified using `--loadqueries`. Neither of these options has a default value.

To produce output in HTML format, use the `--format` option and redirect the output to a file, as shown here:

```
$> ndb_size.pl --database=test --socket=/tmp/mysql.sock --format=html > ndb_size.html
```

(Without the redirection, the output is sent to `stdout`.)

The output from this script includes the following information:

- Minimum values for the `DataMemory`, `IndexMemory`, `MaxNoOfTables`, `MaxNoOfAttributes`, `MaxNoOfOrderedIndexes`, and `MaxNoOfTriggers` configuration parameters required to accommodate the tables analyzed.
- Memory requirements for all of the tables, attributes, ordered indexes, and unique hash indexes defined in the database.
- The `IndexMemory` and `DataMemory` required per table and table row.

## 23.5.29 ndb_top — View CPU usage information for NDB threads

`ndb_top` displays running information in the terminal about CPU usage by NDB threads on an NDB Cluster data node. Each thread is represented by two rows in the output, the first showing system statistics, the second showing the measured statistics for the thread.

`ndb_top` is available beginning with MySQL NDB Cluster 7.6.3.

### Usage

```
ndb_top [-h hostname] [-t port] [-u user] [-p pass] [-n node_id]
```

`ndb_top` connects to a MySQL Server running as an SQL node of the cluster. By default, it attempts to connect to a `mysqld` running on `localhost` and port 3306, as the MySQL `root` user with no password specified. You can override the default host and port using, respectively, `--host` (`-h`) and `--port` (`-t`). To specify a MySQL user and password, use the `--user` (`-u`) and `--passwd` (`-p`) options. This user must be able to read tables in the `ndbinfo` database (`ndb_top` uses information from `ndbinfo.cpustat` and related tables).

For more information about MySQL user accounts and passwords, see Section 6.2, “Access Control and Account Management”.

Output is available as plain text or an ASCII graph; you can specify this using the `--text` (`-x`) and `--graph` (`-g`) options, respectively. These two display modes provide the same information; they can be used concurrently. At least one display mode must be in use.

Color display of the graph is supported and enabled by default (`--color` or `-c` option). With color support enabled, the graph display shows OS user time in blue, OS system time in green, and idle time

as blank. For measured load, blue is used for execution time, yellow for send time, red for time spent in send buffer full waits, and blank spaces for idle time. The percentage shown in the graph display is the sum of percentages for all threads which are not idle. Colors are not currently configurable; you can use grayscale instead by using `--skip-color`.

The sorted view (`--sort`, `-r`) is based on the maximum of the measured load and the load reported by the OS. Display of these can be enabled and disabled using the `--measured-load` (`-m`) and `--os-load` (`-o`) options. Display of at least one of these loads must be enabled.

The program tries to obtain statistics from a data node having the node ID given by the `--node-id` (`-n`) option; if unspecified, this is 1. `ndb_top` cannot provide information about other types of nodes.

The view adjusts itself to the height and width of the terminal window; the minimum supported width is 76 characters.

Once started, `ndb_top` runs continuously until forced to exit; you can quit the program using `Ctrl-C`. The display updates once per second; to set a different delay interval, use `--sleep-time` (`-s`).

**Note**

`ndb_top` is available on macOS, Linux, and Solaris. It is not currently supported on Windows platforms.

The following table includes all options that are specific to the NDB Cluster program `ndb_top`. Additional descriptions follow the table.

**Table 23.50 Command-line options used with the program ndb_top**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--color`, `-c` | Show ASCII graphs in color; use --skip-colors to disable | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--graph`, `-g` | Display data using graphs; use --skip-graphs to disable | (Supported in all NDB releases based on MySQL 8.0) |
| `--help` | Show program usage information | (Supported in all NDB releases based on MySQL 8.0) |
| `--host=string`, `-h string` | Host name or IP address of MySQL Server to connect to | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--measured-load`, `-m` | Show measured load by thread | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--node-id=#`, | Watch node having this node ID | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-n #` | | |
| `--os-load`, `-o` | Show load measured by operating system | (Supported in all NDB releases based on MySQL 8.0) |
| `--password=password`, `-p password` | Connect using this password | (Supported in all NDB releases based on MySQL 8.0) |
| `--port=#`, `-P #` (>=7.6.6) | Port number to use when connecting to MySQL Server | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--sleep-time=#`, `-s #` | Time to wait between display refreshes, in seconds | (Supported in all NDB releases based on MySQL 8.0) |
| `--socket=path`, `-S path` | Socket file to use for connection | (Supported in all NDB releases based on MySQL 8.0) |
| `--sort`, `-r` | Sort threads by usage; use --skip-sort to disable | (Supported in all NDB releases based on MySQL 8.0) |
| `--text`, `-t` (>=7.6.6) | Display data using text | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage` | Show program usage information; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--user=name`, `-u name` | Connect as this MySQL user | (Supported in all NDB releases based on MySQL 8.0) |

## Additional Options

- `--color`, `-c`

| Command-Line Format | `--color` |
|---|---|

  Show ASCII graphs in color; use `--skip-colors` to disable.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

  Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
|---|---|
| Type | String |
| Default Value | `[none]` |

  Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--graph`, `-g`

| Command-Line Format | `--graph` |
| --- | --- |

Display data using graphs; use `--skip-graphs` to disable. This option or `--text` must be true; both options may be true.

- `--help`, `-?`

| Command-Line Format | `--help` |
| --- | --- |

Show program usage information.

- `--host[=name]`, `-h`

| Command-Line Format | `--host=string` |
| --- | --- |
| Type | String |
| Default Value | `localhost` |

Host name or IP address of MySQL Server to connect to.

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--measured-load`, `-m`

| Command-Line Format | `--measured-load` |
| --- | --- |

Show measured load by thread. This option or `--os-load` must be true; both options may be true.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--node-id[=#]`, `-n`

| Command-Line Format | `--node-id=#` |
| --- | --- |
| Type | Integer |
| Default Value | `1` |

Watch the data node having this node ID.

- `--os-load`, `-o`

| Command-Line Format | `--os-load` |
|---|---|

  Show load measured by operating system. This option or `--measured-load` must be true; both options may be true.

- `--password[=password]`, `-p`

| Command-Line Format | `--password=password` |
|---|---|
| Type | String |
| Default Value | `NULL` |

  Connect to a MySQL Server using this password and the MySQL user specified by `--user`.

  This password is associated with a MySQL user account only, and is not related in any way to the password used with encrypted `NDB` backups.

- `--port[=#]`, `-P`

| Command-Line Format | `--port=#` |
|---|---|
| Type | Integer |
| Default Value | `3306` |

  Port number to use when connecting to MySQL Server.

  (Formerly, the short form for this option was `-t`, which was repurposed as the short form of `--text`.)

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

  Print program argument list and exit.

- `--sleep-time[=seconds]`, `-s`

| Command-Line Format | `--sleep-time=#` |
|---|---|
| Type | Integer |
| Default Value | `1` |

  Time to wait between display refreshes, in seconds.

- `--socket=path/to/file`, `-S`

| Command-Line Format | `--socket=path` |
|---|---|
| Type | Path name |
| Default Value | `[none]` |

  Use the specified socket file for the connection.

- `--sort`, `-r`

| Command-Line Format | `--sort` |
|---|---|

  Sort threads by usage; use `--skip-sort` to disable.

- `--text`, `-t`

| Command-Line Format | `--text` |
|---|---|

Display data using text. This option or `--graph` must be true; both options may be true.

(The short form for this option was `-x` in previous versions of NDB Cluster, but this is no longer supported.)

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

Display help text and exit; same as `--help`.

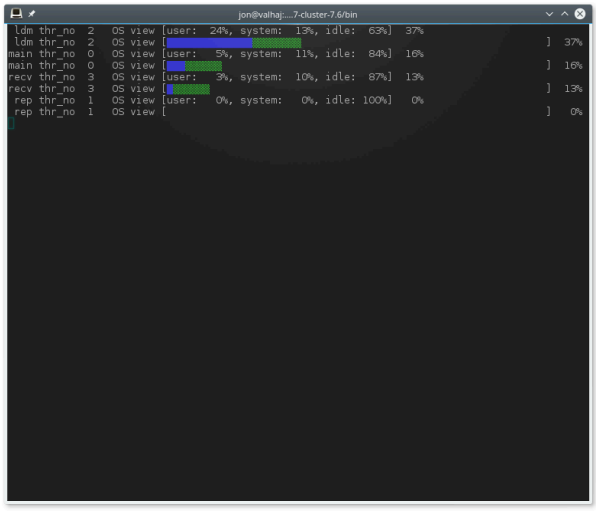
- `--user[=name]`, `-u`

| Command-Line Format | `--user=name` |
|---|---|
| Type | String |
| Default Value | `root` |

Connect as this MySQL user. Normally requires a password supplied by the `--password` option.

**Sample Output.** The next figure shows `ndb_top` running in a terminal window on a Linux system with an `ndbmtd` data node under a moderate load. Here, the program has been invoked using `ndb_top -n8 -x` to provide both text and graph output:

**Figure 23.5 ndb_top Running in Terminal**




Beginning with NDB 8.0.20, `ndb_top` also shows spin times for threads, displayed in green.

## 23.5.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status

`ndb_waiter` repeatedly (each 100 milliseconds) prints out the status of all cluster data nodes until either the cluster reaches a given status or the `--timeout` limit is exceeded, then exits. By default, it waits for the cluster to achieve `STARTED` status, in which all nodes have started and connected to the cluster. This can be overridden using the `--no-contact` and `--not-started` options.

The node states reported by this utility are as follows:

- `NO_CONTACT`: The node cannot be contacted.

- `UNKNOWN`: The node can be contacted, but its status is not yet known. Usually, this means that the node has received a `START` or `RESTART` command from the management server, but has not yet acted on it.
- `NOT_STARTED`: The node has stopped, but remains in contact with the cluster. This is seen when restarting the node using the management client's `RESTART` command.
- `STARTING`: The node's `ndbd` process has started, but the node has not yet joined the cluster.
- `STARTED`: The node is operational, and has joined the cluster.
- `SHUTTING_DOWN`: The node is shutting down.
- `SINGLE USER MODE`: This is shown for all cluster data nodes when the cluster is in single user mode.

Options that can be used with `ndb_waiter` are shown in the following table. Additional descriptions follow the table.

**Table 23.51 Command-line options used with the program ndb_waiter**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--character-sets-dir=path` | Directory containing character sets | REMOVED: 8.0.31 |
| `--connect-retries=#` | Number of times to retry connection before giving up | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-retry-delay=#` | Number of seconds to wait between attempts to contact management server | (Supported in all NDB releases based on MySQL 8.0) |
| `--connect-string=connection_string`, `-c connection_string` | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |
| `--core-file` | Write core file on error; used in debugging | REMOVED: 8.0.31 |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--help`, `-?` | Display help text and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-connectstring=connection_string`, `-c connection_string` | Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;] [host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf | (Supported in all NDB releases based on MySQL 8.0) |
| `--ndb-mgmd-host=connection_string`, | Same as --ndb-connectstring | (Supported in all NDB releases based on MySQL 8.0) |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `-c connection_string` | | |
| `--ndb-nodeid=#` | Set node ID for this node, overriding any ID set by --ndb-connectstring | REMOVED: 8.0.31 |
| `--ndb-optimized-node-selection` | Enable optimizations for selection of nodes for transactions. Enabled by default; use --skip-ndb-optimized-node-selection to disable | REMOVED: 8.0.31 |
| `--no-contact`, `-n` | Wait for cluster to reach NO CONTACT state | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--not-started` | Wait for cluster to reach NOT STARTED state | (Supported in all NDB releases based on MySQL 8.0) |
| `--nowait-nodes=list` | List of nodes not to be waited for | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--single-user` | Wait for cluster to enter single user mode | (Supported in all NDB releases based on MySQL 8.0) |
| `--timeout=#`, `-t #` | Wait this many seconds, then exit whether or not cluster has reached desired state | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Display help text and exit; same as --help | (Supported in all NDB releases based on MySQL 8.0) |
| `--version`, `-V` | Display version information and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--wait-nodes=list`, `-w list` | List of nodes to be waited for | (Supported in all NDB releases based on MySQL 8.0) |

## Usage

```
ndb_waiter [-c connection_string]
```

## Additional Options

- `--character-sets-dir`

| Command-Line Format | `--character-sets-dir=path` |
|---|---|
| Removed | 8.0.31 |

Directory containing character sets.

- `--connect-retries`

| Command-Line Format | `--connect-retries=#` |
|---|---|

| Type | Integer |
| --- | --- |
| Default Value | `12` |
| Minimum Value | `0` |
| Maximum Value | `12` |

Number of times to retry connection before giving up.

- `--connect-retry-delay`

| Command-Line Format | `--connect-retry-delay=#` |
| --- | --- |
| Type | Integer |
| Default Value | `5` |
| Minimum Value | `0` |
| Maximum Value | `5` |

Number of seconds to wait between attempts to contact management server.

- `--connect-string`

| Command-Line Format | `--connect-string=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as `--ndb-connectstring`.

- `--core-file`

| Command-Line Format | `--core-file` |
| --- | --- |
| Removed | 8.0.31 |

Write core file on error; used in debugging.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with concat(group, suffix).

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--help`

| Command-Line Format | `--help` |
| --- | --- |

Display help text and exit.

- `--ndb-connectstring`

| Command-Line Format | `--ndb-connectstring=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Set connect string for connecting to ndb_mgmd. Syntax: "[nodeid=id;][host=]hostname[:port]". Overrides entries in NDB_CONNECTSTRING and my.cnf.

- `--ndb-mgmd-host`

| Command-Line Format | `--ndb-mgmd-host=connection_string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Same as --`ndb-connectstring`.

- `--ndb-nodeid`

| Command-Line Format | `--ndb-nodeid=#` |
| --- | --- |
| Removed | 8.0.31 |
| Type | Integer |
| Default Value | `[none]` |

Set node ID for this node, overriding any ID set by `--ndb-connectstring`.

- `--ndb-optimized-node-selection`

| Command-Line Format | `--ndb-optimized-node-selection` |
| --- | --- |
| Removed | 8.0.31 |

Enable optimizations for selection of nodes for transactions. Enabled by default; use `--skip-ndb-optimized-node-selection` to disable.

- `--no-contact`, `-n`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NO_CONTACT` status before exiting.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
|---|---|

Do not read default options from any option file other than login file.

- `--not-started`

Instead of waiting for the `STARTED` state, `ndb_waiter` continues running until the cluster reaches `NOT_STARTED` status before exiting.

- `--nowait-nodes=list`

When this option is used, `ndb_waiter` does not wait for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

```
$> ndb_waiter --nowait-nodes=1,3,7-9
```

**Important**

Do *not* use this option together with the `--wait-nodes` option.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
|---|---|

Print program argument list and exit.

- `--timeout=seconds`, `-t seconds`

Time to wait. The program exits if the desired state is not achieved within this number of seconds. The default is 120 seconds (1200 reporting cycles).

- `--single-user`

The program waits for the cluster to enter single user mode.

- `--usage`

| Command-Line Format | `--usage` |
|---|---|

Display help text and exit; same as `--help`.

- `--version`

| Command-Line Format | `--version` |
|---|---|

Display version information and exit.

- `--wait-nodes=list`, `-w list`

When this option is used, `ndb_waiter` waits only for the nodes whose IDs are listed. The list is comma-delimited; ranges can be indicated by dashes, as shown here:

```
$> ndb_waiter --wait-nodes=2,4-6,10
```

**Important**

Do *not* use this option together with the `--nowait-nodes` option.

**Sample Output.** Shown here is the output from `ndb_waiter` when run against a 4-node cluster in which two nodes have been shut down and then started again manually. Duplicate reports (indicated by `...`) are omitted.

```
$> ./ndb_waiter -c localhost

Connecting to mgmsrv at (localhost)
State node 1 STARTED
State node 2 NO_CONTACT
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 UNKNOWN
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 NO_CONTACT
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 UNKNOWN
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTING
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTING
Waiting for cluster enter state STARTED

...

State node 1 STARTED
State node 2 STARTED
State node 3 STARTED
State node 4 STARTED
Waiting for cluster enter state STARTED
```

**Note**

If no connection string is specified, then `ndb_waiter` tries to connect to a management on `localhost`, and reports `Connecting to mgmsrv at (null)`.

Prior to NDB 8.0.20, this program printed `NDBT_ProgramExit - status` upon completion of its run, due to an unnecessary dependency on the `NDBT` testing library. This dependency has been removed, eliminating the extraneous output.

## 23.5.31 ndbxfrm — Compress, Decompress, Encrypt, and Decrypt Files Created by NDB Cluster

The `ndbxfrm` utility, introduced in NDB 8.0.22, can be used to decompress, decrypt, and output information about files created by NDB Cluster that are compressed, encrypted, or both. It can also be used to compress or encrypt files.

**Table 23.52 Command-line options used with the program ndbxfrm**

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--compress`, `-c` | Compress file | ADDED: NDB 8.0.22 |
| `--decrypt-key=key` | Supply file decryption key | ADDED: NDB 8.0.31 |
| `--decrypt-key-from-stdin` | Supply file decryption key from stdin | ADDED: NDB 8.0.31 |
| `--decrypt-password=password` | Use this password to decrypt file | ADDED: NDB 8.0.22 |
| `--decrypt-password-from-stdin` | Get decryption password in a secure fashion from STDIN | ADDED: NDB 8.0.24 |
| `--defaults-extra-file=path` | Read given file after global files are read | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-group-suffix=string` | Also read groups with concat(group, suffix) | (Supported in all NDB releases based on MySQL 8.0) |
| `--defaults-file=path` | Read default options from given file only | (Supported in all NDB releases based on MySQL 8.0) |
| `--encrypt-block-size=#` | Print info about file including file header and trailer | ADDED: NDB 8.0.31 |
| `--encrypt-block-size=#` | Size of input data chunks encrypted as a unit. Used with XTS, set to zero for CBC mode | ADDED: NDB 8.0.29 |
| `--encrypt-cipher=#` | Encryption cipher: 1 for CBC, 2 for XTS | ADDED: NDB 8.0.29 |
| `--encrypt-kdf-iter-count=#`, `-k #` | Number of iterations used in key definition | ADDED: NDB 8.0.22 |
| `--encrypt-key=key` | Use this key to encrypt file | ADDED: NDB 8.0.31 |
| `--encrypt-key-from-stdin` | Use key supplied from stdin to encrypt file | ADDED: NDB 8.0.31 |
| `--encrypt-password=password` | Use this password to encrypt file | ADDED: NDB 8.0.22 |
| `--encrypt-password-from-stdin` | Get encryption password in a secure fashion from STDIN | ADDED: NDB 8.0.24 |
| `--help`, `-?` | Print usage information | ADDED: NDB 8.0.22 |
| `--info`, `-i` | Print file information | ADDED: NDB 8.0.22 |

| Format | Description | Added, Deprecated, or Removed |
|---|---|---|
| `--login-path=path` | Read given path from login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--no-defaults` | Do not read default options from any option file other than login file | (Supported in all NDB releases based on MySQL 8.0) |
| `--print-defaults` | Print program argument list and exit | (Supported in all NDB releases based on MySQL 8.0) |
| `--usage`, `-?` | Prints usage information; synonym for --help | ADDED: NDB 8.0.22 |
| `--version`, `-V` | Output version information | ADDED: NDB 8.0.22 |

## Usage

```
ndbxfrm --info file[ file ...]

ndbxfrm --compress input_file output_file

ndbxfrm --decrypt-password=password input_file output_file

ndbxfrm [--encrypt-ldf-iter-count=#] --encrypt-password=password input_file output_file
```

*`input_file`* and *`output_file`* cannot be the same file.

## Options

- `--compress`, `-c`

| Command-Line Format | `--compress` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |

Compresses the input file, using the same compression method as is used for compressing NDB Cluster backups, and writes the output to an output file. To decompress a compressed `NDB` backup file that is not encrypted, it is necessary only to invoke `ndbxfrm` using the names of the compressed file and an output file (with no options required).

- `--decrypt-key=key`, `-K key`

| Command-Line Format | `--decrypt-key=key` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Decrypts a file encrypted by `NDB` using the supplied key.

**Note**

This option cannot be used together with `--decrypt-password`.

- `--decrypt-key-from-stdin`

| Command-Line Format | `--decrypt-key-from-stdin` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Decrypts a file encrypted by `NDB` using the key supplied from `stdin`.

- `--decrypt-password=password`

| Command-Line Format | `--decrypt-password=password` |
| --- | --- |
| Introduced | 8.0.22-ndb-8.0.22 |
| Type | String |
| Default Value | `[none]` |

Decrypts a file encrypted by `NDB` using the password supplied.

**Note**

This option cannot be used together with `--decrypt-key`.

- `--decrypt-password-from-stdin[=TRUE|FALSE]`

| Command-Line Format | `--decrypt-password-from-stdin` |
| --- | --- |
| Introduced | 8.0.24-ndb-8.0.24 |

Decrypts a file encrypted by `NDB`, using a password supplied from standard input. This is similar to entering a password after invoking `mysql --password` with no password following the option.

- `--defaults-extra-file`

| Command-Line Format | `--defaults-extra-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given file after global files are read.

- `--defaults-file`

| Command-Line Format | `--defaults-file=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read default options from given file only.

- `--defaults-group-suffix`

| Command-Line Format | `--defaults-group-suffix=string` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Also read groups with `CONCAT(group, suffix)`.

- `--detailed-info`

| Command-Line Format | `--encrypt-block-size=#` |
| --- | --- |
| Introduced | 8.0.31-ndb-8.0.31 |
| Type | Boolean |

| Default Value | FALSE |
| --- | --- |

Print out file information like `--info`, but include the file's header and trailer.

Example:

```
$> ndbxfrm --detailed-info S0.sysfile
File=/var/lib/cluster-data/ndb_7_fs/D1/NDBCNTR/S0.sysfile, compression=no, encryption=yes
header: {
  fixed_header: {
    magic: {
      magic: { 78, 68, 66, 88, 70, 82, 77, 49 },
      endian: 18364758544493064720,
      header_size: 32768,
      fixed_header_size: 160,
      zeros: { 0, 0 }
    },
    flags: 73728,
    flag_extended: 0,
    flag_zeros: 0,
    flag_file_checksum: 0,
    flag_data_checksum: 0,
    flag_compress: 0,
    flag_compress_method: 0,
    flag_compress_padding: 0,
    flag_encrypt: 18,
    flag_encrypt_cipher: 2,
    flag_encrypt_krm: 1,
    flag_encrypt_padding: 0,
    flag_encrypt_key_selection_mode: 0,
    dbg_writer_ndb_version: 524320,
    octets_size: 32,
    file_block_size: 32768,
    trailer_max_size: 80,
    file_checksum: { 0, 0, 0, 0 },
    data_checksum: { 0, 0, 0, 0 },
    zeros01: { 0 },
    compress_dbg_writer_header_version: { ... },
    compress_dbg_writer_library_version: { ... },
    encrypt_dbg_writer_header_version: { ... },
    encrypt_dbg_writer_library_version: { ... },
    encrypt_key_definition_iterator_count: 100000,
    encrypt_krm_keying_material_size: 32,
    encrypt_krm_keying_material_count: 1,
    encrypt_key_data_unit_size: 32768,
    encrypt_krm_keying_material_position_in_octets: 0,
  },
  octets: {
     102, 68, 56, 125, 78, 217, 110, 94, 145, 121, 203, 234, 26, 164, 137, 180,
     100, 224, 7, 88, 173, 123, 209, 110, 185, 227, 85, 174, 109, 123, 96, 156,
  }
}
trailer: {
  fixed_trailer: {
    flags: 48,
    flag_extended: 0,
    flag_zeros: 0,
    flag_file_checksum: 0,
    flag_data_checksum: 3,
    data_size: 512,
    file_checksum: { 0, 0, 0, 0 },
    data_checksum: { 226, 223, 102, 207 },
    magic: {
      zeros: { 0, 0 }
      fixed_trailer_size: 56,
      trailer_size: 32256,
      endian: 18364758544493064720,
      magic: { 78, 68, 66, 88, 70, 82, 77, 49 },
    },
  }
}
```

- `--encrypt-block-size=#`

| Command-Line Format | `--encrypt-block-size=#` |
|---|---|
| Introduced | 8.0.29-ndb-8.0.29 |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `2147483647` |

Size of input data chunks that are encrypted as a unit. Used with XTS; set to `0` (the default) for CBC mode.

- `--encrypt-cipher=#`

| Command-Line Format | `--encrypt-cipher=#` |
|---|---|
| Introduced | 8.0.29-ndb-8.0.29 |
| Type | Integer |
| Default Value | `1` |
| Minimum Value | `0` |
| Maximum Value | `2147483647` |

Cipher used for encryption. Set to `1` for CBC mode (the default), or `2` for XTS.

- `--encrypt-kdf-iter-count=#`, `-k #`

| Command-Line Format | `--encrypt-kdf-iter-count=#` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |
| Type | Integer |
| Default Value | `0` |
| Minimum Value | `0` |
| Maximum Value | `2147483647` |

When encrypting a file, specifies the number of iterations to use for the encryption key. Requires the `--encrypt-password` option.

- `--encrypt-key=key`

| Command-Line Format | `--encrypt-key=key` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Encrypts a file using the supplied key.

**Note**

This option cannot be used together with `--encrypt-password`.

- `--encrypt-key-from-stdin`

| Command-Line Format | `--encrypt-key-from-stdin` |
|---|---|
| Introduced | 8.0.31-ndb-8.0.31 |

Encrypt a file using the key supplied from `stdin`.

- `--encrypt-password=password`

| Command-Line Format | `--encrypt-password=password` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |
| Type | String |
| Default Value | `[none]` |

Encrypts the backup file using the password supplied by the option. The password must meet the requirements listed here:

- Uses any of the printable ASCII characters except `!`, `'`, `"`, `$`, `%`, `\`, `` ` ``, and `^`
- Is no more than 256 characters in length
- Is enclosed by single or double quotation marks

**Note**

This option cannot be used together with `--encrypt-key`.

- `--encrypt-password-from-stdin[=TRUE|FALSE]`

| Command-Line Format | `--encrypt-password-from-stdin` |
|---|---|
| Introduced | 8.0.24-ndb-8.0.24 |

Encrypts a file using a password supplied from standard input. This is similar to entering a password is entered after invoking `mysql --password` with no password following the option.

- `--help`, `-?`

| Command-Line Format | `--help` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |

Prints usage information for the program.

- `--info`, `-i`

| Command-Line Format | `--info` |
|---|---|
| Introduced | 8.0.22-ndb-8.0.22 |

Prints the following information about one or more input files:

- The name of the file
- Whether the file is compressed (`compression=yes` or `compression=no`)
- Whether the file is encrypted (`encryption=yes` or `encryption=no`)

Example:

```
$> ndbxfrm -i BACKUP-10-0.5.Data BACKUP-10.5.ctl BACKUP-10.5.log
File=BACKUP-10-0.5.Data, compression=no, encryption=yes
File=BACKUP-10.5.ctl, compression=no, encryption=yes
File=BACKUP-10.5.log, compression=no, encryption=yes
```

Beginning with NDB 8.0.31, you can also see the file's header and trailer using the `--detailed-info` option.

- `--login-path`

| Command-Line Format | `--login-path=path` |
| --- | --- |
| Type | String |
| Default Value | `[none]` |

Read given path from login file.

- `--no-defaults`

| Command-Line Format | `--no-defaults` |
| --- | --- |

Do not read default options from any option file other than login file.

- `--print-defaults`

| Command-Line Format | `--print-defaults` |
| --- | --- |

Print program argument list and exit.

- `--usage`, `-?`

| Command-Line Format | `--usage` |
| --- | --- |
| Introduced | 8.0.22-ndb-8.0.22 |

Synonym for `--help`.

- `--version`, `-V`

| Command-Line Format | `--version` |
| --- | --- |
| Introduced | 8.0.22-ndb-8.0.22 |

Prints out version information.

`ndbxfrm` can encrypt backups created by any version of NDB Cluster. The `.Data`, `.ctl`, and `.log` files comprising the backup must be encrypted separately, and these files must be encrypted separately for each data node. Once encrypted, such backups can be decrypted only by `ndbxfrm`, `ndb_restore`, or `ndb_print_backup` from NDB Cluster 8.0.22 or later.

An encrypted file can be re-encrypted with a new password using the `--encrypt-password` and `--decrypt-password` options together, like this:

```
ndbxfrm --decrypt-password=old --encrypt-password=new input_file output_file
```

In the example just shown, *`old`* and *`new`* are the old and new passwords, respectively; both of these must be quoted. The input file is decrypted and then encrypted as the output file. The input file itself is not changed; if you do not want it to be accessible using the old password, you must remove the input file manually.

# 23.6 Management of NDB Cluster

Managing an NDB Cluster involves a number of tasks, the first of which is to configure and start NDB Cluster. This is covered in Section 23.4, “Configuration of NDB Cluster”, and Section 23.5, “NDB Cluster Programs”.

The next few sections cover the management of a running NDB Cluster.

For information about security issues relating to management and deployment of an NDB Cluster, see Section 23.6.20, “NDB Cluster Security Issues”.

There are essentially two methods of actively managing a running NDB Cluster. The first of these is through the use of commands entered into the management client whereby cluster status can be checked, log levels changed, backups started and stopped, and nodes stopped and started. The second method involves studying the contents of the cluster log `ndb_node_id_cluster.log`; this is usually found in the management server's `DataDir` directory, but this location can be overridden using the `LogDestination` option. (Recall that *`node_id`* represents the unique identifier of the node whose activity is being logged.) The cluster log contains event reports generated by `ndbd`. It is also possible to send cluster log entries to a Unix system log.

Some aspects of the cluster's operation can be also be monitored from an SQL node using the `SHOW ENGINE NDB STATUS` statement.

More detailed information about NDB Cluster operations is available in real time through an SQL interface using the `ndbinfo` database. For more information, see Section 23.6.16, “ndbinfo: The NDB Cluster Information Database”.

NDB statistics counters provide improved monitoring using the `mysql` client. These counters, implemented in the NDB kernel, relate to operations performed by or affecting `Ndb` objects, such as starting, closing, and aborting transactions; primary key and unique key operations; table, range, and pruned scans; blocked threads waiting for various operations to complete; and data and events sent and received by NDB Cluster. The counters are incremented by the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes.

`mysqld` exposes the NDB API statistics counters as system status variables, which can be identified from the prefix common to all of their names (`Ndb_api_`). The values of these variables can be read in the `mysql` client from the output of a `SHOW STATUS` statement, or by querying either the Performance Schema `session_status` or `global_status` table. By comparing the values of the status variables before and after the execution of an SQL statement that acts on `NDB` tables, you can observe the actions taken on the NDB API level that correspond to this statement, which can be beneficial for monitoring and performance tuning of NDB Cluster.

MySQL Cluster Manager provides an advanced command-line interface that simplifies many otherwise complex NDB Cluster management tasks, such as starting, stopping, or restarting an NDB Cluster with a large number of nodes. The MySQL Cluster Manager client also supports commands for getting and setting the values of most node configuration parameters as well as `mysqld` server options and variables relating to NDB Cluster. MySQL Cluster Manager version 1.4.8 provides experimental support for NDB 8.0. See MySQL Cluster Manager 1.4.8 User Manual, for more information.

## 23.6.1 Commands in the NDB Cluster Management Client

In addition to the central configuration file, a cluster may also be controlled through a command-line interface available through the management client `ndb_mgm`. This is the primary administrative interface to a running cluster.

Commands for the event logs are given in Section 23.6.3, “Event Reports Generated in NDB Cluster”; commands for creating backups and restoring from them are provided in Section 23.6.8, “Online Backup of NDB Cluster”.

**Using ndb_mgm with MySQL Cluster Manager.** MySQL Cluster Manager 1.4.8 provides experimental support for NDB 8.0. MySQL Cluster Manager handles starting and stopping processes and tracks their states internally, so it is not necessary to use `ndb_mgm` for these tasks for an NDB Cluster that is under MySQL Cluster Manager control. It is recommended *not* to use the `ndb_mgm` command-line client that comes with the NDB Cluster distribution to perform operations that involve starting or stopping nodes. These include but are not limited to the `START`, `STOP`, `RESTART`, and `SHUTDOWN` commands. For more information, see MySQL Cluster Manager Process Commands.

The management client has the following basic commands. In the listing that follows, *`node_id`* denotes either a data node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `CONNECT` *`connection-string`*

Connects to the management server indicated by the connection string. If the client is already connected to this server, the client reconnects.

- `CREATE NODEGROUP nodeid[, nodeid, ...]`

  Creates a new NDB Cluster node group and causes data nodes to join it.

  This command is used after adding new data nodes online to an NDB Cluster, and causes them to join a new node group and thus to begin participating fully in the cluster. The command takes as its sole parameter a comma-separated list of node IDs—these are the IDs of the nodes just added and started, and that are to join the new node group. The list must contain no duplicate IDs; beginning with NDB 8.0.26, the presence of any duplicates causes the command to return an error. The number of nodes in the list must be the same as the number of nodes in each node group that is already part of the cluster (each NDB Cluster node group must have the same number of nodes). In other words, if the NDB Cluster consists of 2 node groups having 2 data nodes each, then the new node group must also have 2 data nodes.

  The node group ID of the new node group created by this command is determined automatically, and always the next highest unused node group ID in the cluster; it is not possible to set it manually.

  For more information, see Section 23.6.7, “Adding NDB Cluster Data Nodes Online”.

- `DROP NODEGROUP nodegroup_id`

  Drops the NDB Cluster node group with the given *`nodegroup_id`*.

  This command can be used to drop a node group from an NDB Cluster. `DROP NODEGROUP` takes as its sole argument the node group ID of the node group to be dropped.

  `DROP NODEGROUP` acts only to remove the data nodes in the effected node group from that node group. It does not stop data nodes, assign them to a different node group, or remove them from the cluster's configuration. A data node that does not belong to a node group is indicated in the output of the management client `SHOW` command with `no nodegroup` in place of the node group ID, like this (indicated using bold text):

  ```
  id=3    @10.100.2.67  (8.0.34-ndb-8.0.34, no nodegroup)
  ```

  `DROP NODEGROUP` works only when all data nodes in the node group to be dropped are completely empty of any table data and table definitions. Since there is currently no way using `ndb_mgm` or the `mysql` client to remove all data from a specific data node or node group, this means that the command succeeds only in the two following cases:

  1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
  2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

     `TRUNCATE TABLE` does not work for this purpose because this removes only the table data; the data nodes continue to store an `NDBCLUSTER` table's definition until a `DROP TABLE` statement is issued that causes the table metadata to be dropped.

  For more information about `DROP NODEGROUP`, see Section 23.6.7, “Adding NDB Cluster Data Nodes Online”.

- `ENTER SINGLE USER MODE node_id`

  Enters single user mode, whereby only the MySQL server identified by the node ID *`node_id`* is permitted to access the database.

  The `ndb_mgm` client provides a clear acknowledgement that this command has been issued and has taken effect, as shown here:

```
ndb_mgm> ENTER SINGLE USER MODE 100
Single user mode entered
Access is granted for API node 100 only.
```

In addition, the API or SQL node having exclusive access when in single user mode is indicated in the output of the `SHOW` command, like this:

```
ndb_mgm> SHOW
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=5    @127.0.0.1  (mysql-8.0.34 ndb-8.0.34, single user mode, Nodegroup: 0, *)
id=6    @127.0.0.1  (mysql-8.0.34 ndb-8.0.34, single user mode, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=50   @127.0.0.1  (mysql-8.0.34 ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=100  @127.0.0.1  (mysql-8.0.34 ndb-8.0.34, allowed single user)
id=101 (not connected, accepting connect from any host)
```

- `EXIT SINGLE USER MODE`

  Exits single user mode, enabling all SQL nodes (that is, all running `mysqld` processes) to access the database.

  **Note**

  It is possible to use `EXIT SINGLE USER MODE` even when not in single user mode, although the command has no effect in this case.

- `HELP`

  Displays information on all available commands.

- *`node_id`* `NODELOG DEBUG {ON|OFF}`

  Toggles debug logging in the node log, as though the effected data node or nodes had been started with the `--verbose` option. `NODELOG DEBUG ON` starts debug logging; `NODELOG DEBUG OFF` switches debug logging off.

- `PROMPT [`*`prompt`*`]`

  Changes the prompt shown by `ndb_mgm` to the string literal *`prompt`*.

  *`prompt`* should not be quoted (unless you want the prompt to include the quotation marks). Unlike the case with the `mysql` client, special character sequences and escapes are not recognized. If called without an argument, the command resets the prompt to the default value (`ndb_mgm>`).

  Some examples are shown here:

  ```
  ndb_mgm> PROMPT mgm#1:
  mgm#1: SHOW
  Cluster Configuration
  ...
  mgm#1: PROMPT mymgm >
  mymgm > PROMPT 'mymgm:'
  'mymgm:' PROMPT  mymgm:
  mymgm: PROMPT
  ndb_mgm> EXIT
  $>
  ```

  Note that leading spaces and spaces within the *`prompt`* string are not trimmed. Trailing spaces are removed.

- `QUIT`, `EXIT`

Terminates the management client.

This command does not affect any nodes connected to the cluster.

- *`node_id`* `REPORT` *`report-type`*

  Displays a report of type *`report-type`* for the data node identified by *`node_id`*, or for all data nodes using `ALL`.

  Currently, there are three accepted values for *`report-type`*:

  - `BackupStatus` provides a status report on a cluster backup in progress
  - `MemoryUsage` displays how much data memory and index memory is being used by each data node as shown in this example:

    ```
    ndb_mgm> ALL REPORT MEMORY

    Node 1: Data usage is 5%(177 32K pages of total 3200)
    Node 1: Index usage is 0%(108 8K pages of total 12832)
    Node 2: Data usage is 5%(177 32K pages of total 3200)
    Node 2: Index usage is 0%(108 8K pages of total 12832)
    ```

    This information is also available from the `ndbinfo.memoryusage` table.

  - `EventLog` reports events from the event log buffers of one or more data nodes.

  *`report-type`* is case-insensitive and “fuzzy”; for `MemoryUsage`, you can use `MEMORY` (as shown in the prior example), `memory`, or even simply `MEM` (or `mem`). You can abbreviate `BackupStatus` in a similar fashion.

- *`node_id`* `RESTART [-n] [-i] [-a] [-f]`

  Restarts the data node identified by *`node_id`* (or all data nodes).

  Using the `-i` option with `RESTART` causes the data node to perform an initial restart; that is, the node's file system is deleted and recreated. The effect is the same as that obtained from stopping the data node process and then starting it again using `ndbd --initial` from the system shell.

  **Note**

  Backup files and Disk Data files are not removed when this option is used.

  Using the `-n` option causes the data node process to be restarted, but the data node is not actually brought online until the appropriate `START` command is issued. The effect of this option is the same as that obtained from stopping the data node and then starting it again using `ndbd --nostart` or `ndbd -n` from the system shell.

  Using the `-a` causes all current transactions relying on this node to be aborted. No GCP check is done when the node rejoins the cluster.

  Normally, `RESTART` fails if taking the node offline would result in an incomplete cluster. The `-f` option forces the node to restart without checking for this. If this option is used and the result is an incomplete cluster, the entire cluster is restarted.

- `SHOW`

  Displays basic information about the cluster and cluster nodes. For all nodes, the output includes the node's ID, type, and `NDB` software version. If the node is connected, its IP address is also shown;

otherwise the output shows `not connected, accepting connect from ip_address`, with `any host` used for nodes that are permitted to connect from any address.

In addition, for data nodes, the output includes `starting` if the node has not yet started, and shows the node group of which the node is a member. If the data node is acting as the master node, this is indicated with an asterisk (`*`).

Consider a cluster whose configuration file includes the information shown here (possible additional settings are omitted for clarity):

```
[ndbd default]
DataMemory= 128G
NoOfReplicas= 2

[ndb_mgmd]
NodeId=50
HostName=198.51.100.150

[ndbd]
NodeId=5
HostName=198.51.100.10
DataDir=/var/lib/mysql-cluster

[ndbd]
NodeId=6
HostName=198.51.100.20
DataDir=/var/lib/mysql-cluster

[ndbd]
NodeId=7
HostName=198.51.100.30
DataDir=/var/lib/mysql-cluster

[ndbd]
NodeId=8
HostName=198.51.100.40
DataDir=/var/lib/mysql-cluster

[mysqld]
NodeId=100
HostName=198.51.100.100

[api]
NodeId=101
```

After this cluster (including one SQL node) has been started, `SHOW` displays the following output:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     4 node(s)
id=5    @198.51.100.10  (mysql-8.0.34 ndb-8.0.34, Nodegroup: 0, *)
id=6    @198.51.100.20  (mysql-8.0.34 ndb-8.0.34, Nodegroup: 0)
id=7    @198.51.100.30  (mysql-8.0.34 ndb-8.0.34, Nodegroup: 1)
id=8    @198.51.100.40  (mysql-8.0.34 ndb-8.0.34, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=50   @198.51.100.150  (mysql-8.0.34 ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=100  @198.51.100.100  (mysql-8.0.34 ndb-8.0.34)
id=101 (not connected, accepting connect from any host)
```

The output from this command also indicates when the cluster is in single user mode (see the description of the `ENTER SINGLE USER MODE` command, as well as Section 23.6.6, “NDB Cluster Single User Mode”). In NDB 8.0, it also indicates which API or SQL node has exclusive access when this mode is in effect; this works only when all data nodes and management nodes connected to the cluster are running NDB 8.0.

- `SHUTDOWN`

  Shuts down all cluster data nodes and management nodes. To exit the management client after this has been done, use `EXIT` or `QUIT`.

  This command does *not* shut down any SQL nodes or API nodes that are connected to the cluster.

- *`node_id`* `START`

  Brings online the data node identified by *`node_id`* (or all data nodes).

  `ALL START` works on all data nodes only, and does not affect management nodes.

  **Important**

  To use this command to bring a data node online, the data node must have been started using `--nostart` or `-n`.

- *`node_id`* `STATUS`

  Displays status information for the data node identified by *`node_id`* (or for all data nodes).

  Possible node status values include `UNKNOWN`, `NO_CONTACT`, `NOT_STARTED`, `STARTING`, `STARTED`, `SHUTTING_DOWN`, and `RESTARTING`.

  The output from this command also indicates when the cluster is in single user mode.

- *`node_id`* `STOP [-a] [-f]`

  Stops the data or management node identified by *`node_id`*.

  **Note**

  `ALL STOP` works to stop all data nodes only, and does not affect management nodes.

  A node affected by this command disconnects from the cluster, and its associated `ndbd` or `ndb_mgmd` process terminates.

  The `-a` option causes the node to be stopped immediately, without waiting for the completion of any pending transactions.

  Normally, `STOP` fails if the result would cause an incomplete cluster. The `-f` option forces the node to shut down without checking for this. If this option is used and the result is an incomplete cluster, the cluster immediately shuts down.

  **Warning**

  Use of the `-a` option also disables the safety check otherwise performed when `STOP` is invoked to insure that stopping the node does not cause an incomplete cluster. In other words, you should exercise extreme care when using the `-a` option with the `STOP` command, due to the fact that this option makes it possible for the cluster to undergo a forced shutdown because it no longer has a complete copy of all data stored in `NDB`.

**Additional commands.** A number of other commands available in the `ndb_mgm` client are described elsewhere, as shown in the following list:

- `START BACKUP` is used to perform an online backup in the `ndb_mgm` client; the `ABORT BACKUP` command is used to cancel a backup already in progress. For more information, see Section 23.6.8, “Online Backup of NDB Cluster”.

- The `CLUSTERLOG` command is used to perform various logging functions. See Section 23.6.3, “Event Reports Generated in NDB Cluster”, for more information and examples. `NODELOG DEBUG` activates or deactivates debug printouts in node logs, as described previously in this section.
- For testing and diagnostics work, the client supports a `DUMP` command which can be used to execute internal commands on the cluster. It should never be used in a production setting unless directed to do so by MySQL Support. For more information, see NDB Cluster Management Client DUMP Commands.

## 23.6.2 NDB Cluster Log Messages

This section contains information about the messages written to the cluster log in response to different cluster log events. It provides additional, more specific information on `NDB` transporter errors.

### 23.6.2.1 NDB Cluster: Messages in the Cluster Log

The following table lists the most common `NDB` cluster log messages. For information about the cluster log, log events, and event types, see Section 23.6.3, “Event Reports Generated in NDB Cluster”. These log messages also correspond to log event types in the MGM API; see The Ndb_logevent_type Type, for related information of interest to Cluster API developers.

**Table 23.53 Common NDB cluster log messages**

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `Node mgm_node_id: Node data_node_id Connected` | The data node having node ID *node_id* has connected to the management server (node *mgm_node_id*). | `Connected` | `Connection` | 8 | `INFO` |
| `Node mgm_node_id: Node data_node_id Disconnected` | The data node having node ID *data_node_id* has disconnected from the management server (node *mgm_node_id*). | `Disconnected` | `Connection` | 8 | `ALERT` |
| `Node data_node_id: Communication to Node api_node_id closed` | The API node or SQL node having node ID *api_node_id* is no longer communicating with data node *data_node_id*. | `CommunicationClosed` | `Connection` | 8 | `INFO` |
| `Node data_node_id: Communication to Node api_node_id opened` | The API node or SQL node having node ID *api_node_id* is now communicating with data node *data_node_id*. | `CommunicationOpened` | `Connection` | 8 | `INFO` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| Node *mgm_node_id*: Node *api_node_id*: API *version* | The API node having node ID *api_node_id* has connected to management node *mgm_node_id* using NDB API version *version* (generally the same as the MySQL version number). | ConnectedApiVersion | Connection | 8 | INFO |
| Node *node_id*: Global checkpoint *gci* started | A global checkpoint with the ID *gci* has been started; node *node_id* is the master responsible for this global checkpoint. | GlobalCheckpointStarted | Checkpoint | 9 | INFO |
| Node *node_id*: Global checkpoint *gci* completed | The global checkpoint having the ID *gci* has been completed; node *node_id* was the master responsible for this global checkpoint. | GlobalCheckpointCompleted | Checkpoint | 10 | INFO |
| Node *node_id*: Local checkpoint *lcp* started. Keep GCI = *current_gci* oldest restorable GCI = *old_gci* | The local checkpoint having sequence ID *lcp* has been started on node *node_id*. The most recent GCI that can be used has the index *current_gci*, and the oldest GCI from which the cluster can be restored has the index *old_gci*. | LocalCheckpointStarted | Checkpoint | 7 | INFO |
| Node *node_id*: Local checkpoint | The local checkpoint having sequence ID | LocalCheckpointCompleted | Checkpoint | 8 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| *lcp* completed | *lcp* on node *node_id* has been completed. | | | | |
| Node *node_id*: Local Checkpoint stopped in CALCULATED_KEEP_GCI | The node was unable to determine the most recent usable GCI. | LCPStoppedInCalculatedKeepGci | Checkpoint | 0 | ALERT |
| Node *node_id*: Table ID = *table_id*, fragment ID = *fragment_id* has completed LCP on Node *node_id* maxGciStarted: *started_gci* maxGciCompleted: *completed_gci* | A table fragment has been checkpointed to disk on node *node_id*. The GCI in progress has the index *started_gci*, and the most recent GCI to have been completed has the index *completed_gci*. | LCPFragmentCompleted | Checkpoint | 11 | INFO |
| Node *node_id*: ACC Blocked *num_1* and TUP Blocked *num_2* times last second | Undo logging is blocked because the log buffer is close to overflowing. | UndoLogBlocked | Checkpoint | 7 | INFO |
| Node *node_id*: Start initiated *version* | Data node *node_id*, running NDB version *version*, is beginning its startup process. | NDBStartStarted | StartUp | 1 | INFO |
| Node *node_id*: Started *version* | Data node *node_id*, running NDB version *version*, has started successfully. | NDBStartCompleted | StartUp | 1 | INFO |
| Node *node_id*: STTORRY received after restart finished | The node has received a signal indicating that a cluster restart has completed. | STTORRYRecieved | StartUp | 15 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `Node node_id: Start phase phase completed (type)` | The node has completed start phase *phase* of a *type* start. For a listing of start phases, see Section 23.6.4, “Summary of NDB Cluster Start Phases”. (*type* is one of `initial`, `system`, `node`, `initial node`, or `<Unknown>`.) | `StartPhaseCompleted` | `StartUp` | 4 | `INFO` |
| `Node node_id: CM_REGCONF president = president_id, own Node = own_id, our dynamic id = dynamic_id` | Node *president_id* has been selected as “president”. *own_id* and *dynamic_id* should always be the same as the ID (*node_id*) of the reporting node. | `CM_REGCONF` | `StartUp` | 3 | `INFO` |
| `Node node_id: CM_REGREF from Node president_id to our Node node_id. Cause = cause` | The reporting node (ID *node_id*) was unable to accept node *president_id* as president. The *cause* of the problem is given as one of `Busy`, `Election with wait = false`, `Not president`, `Election without selecting new candidate`, or `No such cause`. | `CM_REGREF` | `StartUp` | 8 | `INFO` |
| `Node node_id: We are Node` | The node has discovered its neighboring | `FIND_NEIGHBOURS` | `StartUp` | 8 | `INFO` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| *own_id* with dynamic ID *dynamic_id*, our left neighbor is Node *id_1*, our right is Node *id_2* | nodes in the cluster (node *id_1* and node *id_2*). *node_id*, *own_id*, and *dynamic_id* should always be the same; if they are not, this indicates a serious misconfiguration of the cluster nodes. | | | | |
| Node *node_id*: *type* shutdown initiated | The node has received a shutdown signal. The *type* of shutdown is either Cluster or Node. | NDBStopStarted | StartUp | 1 | INFO |
| Node *node_id*: Node shutdown completed [, *action*] [Initiated by signal *signal*.] | The node has been shut down. This report may include an *action*, which if present is one of restarting, no start, or initial. The report may also include a reference to an NDB Protocol *signal*; for possible signals, refer to Operations and Signals. | NDBStopCompleted | StartUp | 1 | INFO |
| Node *node_id*: Forced node shutdown completed [, action]. [Occurred during startphase *start_phase*.] [ Initiated by *signal*.] | The node has been forcibly shut down. The *action* (one of restarting, no start, or initial) subsequently being taken, if any, is also reported. If the shutdown | NDBStopForced | StartUp | 1 | ALERT |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| [Caused by error *error_code*: '*error_message*(*error_classification*). *error_status*' [(extra info *extra_code*)]] | occurred while the node was starting, the report includes the *start_phase* during which the node failed. If this was a result of a *signal* sent to the node, this information is also provided (see Operations and Signals, for more information). If the error causing the failure is known, this is also included; for more information about NDB error messages and classifications, see NDB Cluster API Errors. | | | | |
| Node *node_id*: Node shutdown aborted | The node shutdown process was aborted by the user. | NDBStopAborted | StartUp | 1 | INFO |
| Node *node_id*: StartLog: [GCI Keep: *keep_pos* LastCompleted *last_pos* NewestRestorable *restore_pos*] | This reports global checkpoints referenced during a node start. The redo log prior to *keep_pos* is dropped. *last_pos* is the last global checkpoint in which data node the participated; *restore_pos* is the global checkpoint which is actually used to | StartREDOLog | StartUp | 4 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| | restore all data nodes. | | | | |
| *startup_message* [*Listed separately; see below.*] | There are a number of possible startup messages that can be logged under different circumstances. These are listed separately; see Section 23.6.2.2, “NDB Cluster Log Startup Messages”. | StartReport | StartUp | 4 | INFO |
| Node *node_id*: Node restart completed copy of dictionary information | Copying of data dictionary information to the restarted node has been completed. | NR_CopyDict | NodeRestart | 8 | INFO |
| Node *node_id*: Node restart completed copy of distribution information | Copying of data distribution information to the restarted node has been completed. | NR_CopyDistr | NodeRestart | 8 | INFO |
| Node *node_id*: Node restart starting to copy the fragments to Node *node_id* | Copy of fragments to starting data node *node_id* has begun | NR_CopyFragsStarted | NodeRestart | 8 | INFO |
| Node *node_id*: Table ID = *table_id*, fragment ID = *fragment_id* have been copied to Node *node_id* | Fragment *fragment_id* from table *table_id* has been copied to data node *node_id* | NR_CopyFragDone | NodeRestart | 10 | INFO |
| Node *node_id*: Node | Copying of all table fragments to restarting | NR_CopyFragsCompleted | NodeRestart | 8 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `restart completed copying the fragments to Node node_id` | data node *node_id* has been completed | | | | |
| `Node node_id: Node node1_id completed failure of Node node2_id` | Data node *node1_id* has detected the failure of data node *node2_id* | `NodeFailCompleted` | `NodeRestart` | 8 | `ALERT` |
| `All nodes completed failure of Node node_id` | All (remaining) data nodes have detected the failure of data node *node_id* | `NodeFailCompleted` | `NodeRestart` | 8 | `ALERT` |
| `Node failure of node_idblock completed` | The failure of data node *node_id* has been detected in the *block*`NDB` kernel block, where block is 1 of `DBTC`, `DBDICT`, `DBDIH`, or `DBLQH`; for more information, see NDB Kernel Blocks | `NodeFailCompleted` | `NodeRestart` | 8 | `ALERT` |
| `Node mgm_node_id: Node data_node_id has failed. The Node state at failure was state_code` | A data node has failed. Its state at the time of failure is described by an arbitration state code *state_code*: possible state code values can be found in the file `include/kernel/signaldata/ArbitSignalData.hpp`. | `NODE_FAILREP` | `NodeRestart` | 8 | `ALERT` |
| `President restarts` | This is a report on the current | `ArbitState` | `NodeRestart` | 6 | `INFO` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `arbitration thread [state=`*`state_code`*`]` or `Prepare arbitrator node` *`node_id`* `[ticket=`*`ticket_id`*`]` or `Receive arbitrator node` *`node_id`* `[ticket=`*`ticket_id`*`]` or `Started arbitrator node` *`node_id`* `[ticket=`*`ticket_id`*`]` or `Lost arbitrator node` *`node_id`* `- process failure [state=`*`state_code`*`]` or `Lost arbitrator node` *`node_id`* `- process exit [state=`*`state_code`*`]` or `Lost arbitrator node` *`node_id`* `- `*`error_message`*` [state=`*`state_code`*`]` | state and progress of arbitration in the cluster. *`node_id`* is the node ID of the management node or SQL node selected as the arbitrator. *`state_code`* is an arbitration state code, as found in `include/kernel/signaldata/ArbitSignalData.hpp`. When an error has occurred, an *`error_message`*, also defined in `ArbitSignalData.hpp`, is provided. *`ticket_id`* is a unique identifier handed out by the arbitrator when it is selected to all the nodes that participated in its selection; this is used to ensure that each node requesting arbitration was one of the nodes that took part in the selection process. | | | | |
| `Arbitration check lost - less than 1/2 nodes left` or `Arbitration check won - all node groups and more than` | This message reports on the result of arbitration. In the event of arbitration failure, an *`error_message`* and an arbitration | `ArbitResult` | `NodeRestart` | 2 | `ALERT` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `1/2 nodes left` or `Arbitration check won - node group majority` or `Arbitration check lost - missing node group` or `Network partitioning - arbitration required` or `Arbitration won - positive reply from node` *`node_id`* or `Arbitration lost - negative reply from node` *`node_id`* or `Network partitioning - no arbitrator available` or `Network partitioning - no arbitrator configured` or `Arbitration failure -` *`error_message`* `[state=`*`state_code`*`]` | *`state_code`* are provided; definitions for both of these are found in `include/kernel/signaldata/ArbitSignalData.hpp`. | | | | |
| `Node` *`node_id`*`: GCP Take over started` | This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node) | `GCP_TakeoverStarted` | `NodeRestart` | 7 | `INFO` |
| `Node` *`node_id`*`: GCP Take over completed` | This node has become the master, and has assumed responsibility | `GCP_TakeoverCompleted` | `NodeRestart` | 7 | `INFO` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| | for the next global checkpoint | | | | |
| Node *node_id*: LCP Take over started | This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node) | LCP_TakeoverStarted | NodeRestart | 7 | INFO |
| Node *node_id*: LCP Take over completed | This node has become the master, and has assumed responsibility for the next set of local checkpoints | LCP_TakeoverCompleted | NodeRestart | 7 | INFO |
| Node *node_id*: Trans. Count = *transactions*, Commit Count = *commits*, Read Count = *reads*, Simple Read Count = *simple_reads*, Write Count = *writes*, AttrInfo Count = *AttrInfo_objects*, Concurrent Operations = *concurrent_operations*, Abort Count = *aborts*, Scans = *scans*, Range scans = *range_scans* | This report of transaction activity is given approximately once every 10 seconds | TransReportCounters | Statistic | 8 | INFO |
| Node *node_id*: Operations=*operations* | Number of operations performed by this node, provided | OperationReportCounters | Statistic | 8 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| | approximately once every 10 seconds | | | | |
| Node *node_id*: Table with ID = *table_id* created | A table having the table ID shown has been created | TableCreated | Statistic | 7 | INFO |
| Node *node_id*: Mean loop Counter in doJob last 8192 times = *count* | | JobStatistic | Statistic | 9 | INFO |
| Mean send size to Node = *node_id* last 4096 sends = *bytes* bytes | This node is sending an average of *bytes* bytes per send to node *node_id* | SendBytesStatistic | Statistic | 9 | INFO |
| Mean receive size to Node = *node_id* last 4096 sends = *bytes* bytes | This node is receiving an average of *bytes* of data each time it receives data from node *node_id* | ReceiveBytesStatistic | Statistic | 9 | INFO |
| Node *node_id*: Data usage is *data_memory_percentage*% (*data_pages_used* 32K pages of total *data_pages_total*) / Node *node_id*: Index usage is *index_memory_percentage*% (*index_pages_used* 8K pages of total *index_pages_total*) | This report is generated when a DUMP 1000 command is issued in the cluster management client | MemoryUsage | Statistic | 5 | INFO |
| Node *node1_id*: Transporter to node *node2_id* | A transporter error occurred while communicating with node | TransporterError | Error | 2 | ERROR |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| reported error *error_code*: *error_message* | *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL NDB Cluster Internals Manual | | | | |
| Node *node1_id*: Transporter to node *node2_id* reported error *error_code*: *error_message* | A warning of a potential transporter problem while communicating with node *node2_id*; for a listing of transporter error codes and messages, see NDB Transporter Errors, for more information | TransporterWarning | Error | 8 | WARNING |
| Node *node1_id*: Node *node2_id* missed heartbeat *heartbeat_id* | This node missed a heartbeat from node *node2_id* | MissedHeartbeat | Error | 8 | WARNING |
| Node *node1_id*: Node *node2_id* declared dead due to missed heartbeat | This node has missed at least 3 heartbeats from node *node2_id*, and so has declared that node “dead” | DeadDueToHeartbeat | Error | 8 | ALERT |
| Node *node1_id*: Node Sent Heartbeat to node = *node2_id* | This node has sent a heartbeat to node *node2_id* | SentHeartbeat | Info | 12 | INFO |
| Node *node_id*: Event buffer status (*object_id*): | This report is seen during heavy event buffer usage, for example, when many | EventBufferStatus2 | Info | 7 | INFO |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `used=bytes_used (percent_used% of alloc) alloc=bytes_allocated max=bytes_available latest_consumed_epoch=latest_consumed_epoch latest_buffered_epoch=latest_buffered_epoch report_reason=report_reason` | updates are being applied in a relatively short period of time; the report shows the number of bytes and the percentage of event buffer memory used, the bytes allocated and percentage still available, and the latest buffered and consumed epochs; for more information, see Section 23.6.2.3, “Event Buffer Reporting in the Cluster Log” | | | | |
| `Node node_id: Entering single user mode`, `Node node_id: Entered single user mode Node API_node_id has exclusive access`, `Node node_id: Entering single user mode` | These reports are written to the cluster log when entering and exiting single user mode; *API_node_id* is the node ID of the API or SQL having exclusive access to the cluster (for more information, see Section 23.6.6, “NDB Cluster Single User Mode”); the message `Unknown single user report API_node_id` indicates an error has taken place and should never be seen in normal operation | `SingleUser` | `Info` | 7 | `INFO` |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| Node *node_id*: Backup *backup_id* started from node *mgm_node_id* | A backup has been started using the management node having *mgm_node_id*; this message is also displayed in the cluster management client when the START BACKUP command is issued; for more information, see Section 23.6.8.2, "Using The NDB Cluster Management Client to Create a Backup" | BackupStarted | Backup | 7 | INFO |
| Node *node_id*: Backup *backup_id* started from node *mgm_node_id* completed. StartGCP: *start_gcp* StopGCP: *stop_gcp* #Records: *records* #LogRecords: *log_records* Data: *data_bytes* bytes Log: *log_bytes* bytes | The backup having the ID *backup_id* has been completed; for more information, see Section 23.6.8.2, "Using The NDB Cluster Management Client to Create a Backup" | BackupCompleted | Backup | 7 | INFO |
| Node *node_id*: Backup request from *mgm_node_id* failed to start. Error: *error_code* | The backup failed to start; for error codes, see MGM API Errors | BackupFailedToStart | Backup | 7 | ALERT |
| Node *node_id*: | The backup was terminated | BackupAborted | Backup | 7 | ALERT |

| Log Message | Description | Event Name | Event Type | Priority | Severity |
|---|---|---|---|---|---|
| `Backup backup_id started from mgm_node_id has been aborted. Error: error_code` | after starting, possibly due to user intervention | | | | |

### 23.6.2.2 NDB Cluster Log Startup Messages

Possible startup messages with descriptions are provided in the following list:

- `Initial start, waiting for %s to connect, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting until nodes: %s connects, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting %u sec for nodes %s to connect, nodes [ all: %s connected: %s no-wait: %s ]`
- `Waiting for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`
- `Waiting %u sec for non partitioned start, nodes [ all: %s connected: %s missing: %s no-wait: %s ]`
- `Initial start with nodes %s [ missing: %s no-wait: %s ]`
- `Start with all nodes %s`
- `Start with nodes %s [ missing: %s no-wait: %s ]`
- `Start potentially partitioned with nodes %s [ missing: %s no-wait: %s ]`
- `Unknown startreport: 0x%x [ %s %s %s %s ]`

### 23.6.2.3 Event Buffer Reporting in the Cluster Log

`NDB` uses one or more memory buffers for events received from the data nodes. There is one such buffer for each `Ndb` object subscribing to table events, which means that there are usually two buffers for each `mysqld` performing binary logging (one buffer for schema events, and one for data events). Each buffer contains epochs made up of events. These events consist of operation types (insert, update, delete) and row data (before and after images plus metadata).

`NDB` generates messages in the cluster log to describe the state of these buffers. Although these reports appear in the cluster log, they refer to buffers on API nodes (unlike most other cluster log messages, which are generated by data nodes).

Event buffer logging reports in the cluster log use the format shown here:

```
Node node_id: Event buffer status (object_id):
used=bytes_used (percent_used% of alloc)
alloc=bytes_allocated (percent_alloc% of max) max=bytes_available
latest_consumed_epoch=latest_consumed_epoch
latest_buffered_epoch=latest_buffered_epoch
report_reason=report_reason
```

The fields making up this report are listed here, with descriptions:

- *node_id*: ID of the node where the report originated.
- *object_id*: ID of the Ndb object where the report originated.
- *bytes_used*: Number of bytes used by the buffer.
- *percent_used*: Percentage of allocated bytes used.
- *bytes_allocated*: Number of bytes allocated to this buffer.
- *percent_alloc*: Percentage of available bytes used; not printed if ndb_eventbuffer_max_alloc is equal to 0 (unlimited).
- *bytes_available*: Number of bytes available; this is 0 if ndb_eventbuffer_max_alloc is 0 (unlimited).
- *latest_consumed_epoch*: The epoch most recently consumed to completion. (In NDB API applications, this is done by calling nextEvent().)
- *latest_buffered_epoch*: The epoch most recently buffered (completely) in the event buffer.
- *report_reason*: The reason for making the report. Possible reasons are shown later in this section.

Possible reasons for reporting are described in the following list:

- ENOUGH_FREE_EVENTBUFFER: The event buffer has sufficient space.

  LOW_FREE_EVENTBUFFER: The event buffer is running low on free space.

  The threshold free percentage level triggering these reports can be adjusted by setting the ndb_report_thresh_binlog_mem_usage server variable.
- BUFFERED_EPOCHS_OVER_THRESHOLD: Whether the number of buffered epochs has exceeded the configured threshold. This number is the difference between the latest epoch that has been received in its entirety and the epoch that has most recently been consumed (in NDB API applications, this is done by calling nextEvent() or nextEvent2()). The report is generated every second until the number of buffered epochs goes below the threshold, which can be adjusted by setting the ndb_report_thresh_binlog_epoch_slip server variable. You can also adjust the threshold in NDB API applications by calling setEventBufferQueueEmptyEpoch().
- PARTIALLY_DISCARDING: Event buffer memory is exhausted—that is, 100% of ndb_eventbuffer_max_alloc has been used. Any partially buffered epoch is buffered to completion even is usage exceeds 100%, but any new epochs received are discarded. This means that a gap has occurred in the event stream.
- COMPLETELY_DISCARDING: No epochs are buffered.
- PARTIALLY_BUFFERING: The buffer free percentage following the gap has risen to the threshold, which can be set in the mysql client using the ndb_eventbuffer_free_percent server system variable or in NDB API applications by calling set_eventbuffer_free_percent(). New epochs are buffered. Epochs that could not be completed due to the gap are discarded.
- COMPLETELY_BUFFERING: All epochs received are being buffered, which means that there is sufficient event buffer memory. The gap in the event stream has been closed.

### 23.6.2.4 NDB Cluster: NDB Transporter Errors

This section lists error codes, names, and messages that are written to the cluster log in the event of transporter errors.

0x00 **TE_NO_ERROR**

No error

| | |
|---|---|
| 0x01 | **TE_ERROR_CLOSING_SOCKET** |
| | Error found during closing of socket |
| 0x02 | **TE_ERROR_IN_SELECT_BEFORE_ACCEPT** |
| | Error found before accept. The transporter will retry |
| 0x03 | **TE_INVALID_MESSAGE_LENGTH** |
| | Error found in message (invalid message length) |
| 0x04 | **TE_INVALID_CHECKSUM** |
| | Error found in message (checksum) |
| 0x05 | **TE_COULD_NOT_CREATE_SOCKET** |
| | Error found while creating socket(can't create socket) |
| 0x06 | **TE_COULD_NOT_BIND_SOCKET** |
| | Error found while binding server socket |
| 0x07 | **TE_LISTEN_FAILED** |
| | Error found while listening to server socket |
| 0x08 | **TE_ACCEPT_RETURN_ERROR** |
| | Error found during accept(accept return error) |
| 0x0b | **TE_SHM_DISCONNECT** |
| | The remote node has disconnected |
| 0x0c | **TE_SHM_IPC_STAT** |
| | Unable to check shm segment |
| 0x0d | **TE_SHM_UNABLE_TO_CREATE_SEGMENT** |
| | Unable to create shm segment |
| 0x0e | **TE_SHM_UNABLE_TO_ATTACH_SEGMENT** |
| | Unable to attach shm segment |
| 0x0f | **TE_SHM_UNABLE_TO_REMOVE_SEGMENT** |
| | Unable to remove shm segment |
| 0x10 | **TE_TOO_SMALL_SIGID** |
| | Sig ID too small |
| 0x11 | **TE_TOO_LARGE_SIGID** |
| | Sig ID too large |
| 0x12 | **TE_WAIT_STACK_FULL** |
| | Wait stack was full |

| | |
|---|---|
| 0x13 | **TE_RECEIVE_BUFFER_FULL**<br>Receive buffer was full |
| 0x14 | **TE_SIGNAL_LOST_SEND_BUFFER_FULL**<br>Send buffer was full,and trying to force send fails |
| 0x15 | **TE_SIGNAL_LOST**<br>Send failed for unknown reason(signal lost) |
| 0x16 | **TE_SEND_BUFFER_FULL**<br>The send buffer was full, but sleeping for a while solved |
| 0x21 | **TE_SHM_IPC_PERMANENT**<br>Shm ipc Permanent error |

**Note**

Transporter error codes `0x17` through `0x20` and `0x22` are reserved for SCI connections, which are not supported in this version of NDB Cluster, and so are not included here.

## 23.6.3 Event Reports Generated in NDB Cluster

In this section, we discuss the types of event logs provided by NDB Cluster, and the types of events that are logged.

NDB Cluster provides two types of event log:

- The *cluster log*, which includes events generated by all cluster nodes. The cluster log is the log recommended for most uses because it provides logging information for an entire cluster in a single location.

  By default, the cluster log is saved to a file named `ndb_node_id_cluster.log`, (where *`node_id`* is the node ID of the management server) in the management server's `DataDir`.

  Cluster logging information can also be sent to `stdout` or a `syslog` facility in addition to or instead of being saved to a file, as determined by the values set for the `DataDir` and `LogDestination` configuration parameters. See Section 23.4.3.5, “Defining an NDB Cluster Management Server”, for more information about these parameters.

- *Node logs* are local to each node.

  Output generated by node event logging is written to the file `ndb_node_id_out.log` (where *`node_id`* is the node's node ID) in the node's `DataDir`. Node event logs are generated for both management nodes and data nodes.

  Node logs are intended to be used only during application development, or for debugging application code.

Both types of event logs can be set to log different subsets of events.

Each reportable event can be distinguished according to three different criteria:

- *Category*: This can be any one of the following values: `STARTUP`, `SHUTDOWN`, `STATISTICS`, `CHECKPOINT`, `NODERESTART`, `CONNECTION`, `ERROR`, or `INFO`.

- *Priority*: This is represented by one of the numbers from 0 to 15 inclusive, where 0 indicates “most important” and 15 “least important.”
- *Severity Level*: This can be any one of the following values: `ON`, `DEBUG`, `INFO`, `WARNING`, `ERROR`, `CRITICAL`, `ALERT`, or `ALL`. (This is also sometimes referred to as the log level.)

Both the cluster log and the node log can be filtered on these properties.

The format used in a log message generated by NDB Cluster (as of NDB 8.0.26) is as shown here:

```
timestamp [node_type] level -- Node node_id: message
```

Each line in the log, or log message, contains the following information:

- A *`timestamp`* in *`YYYY-MM-DD HH:MM:SS`* format. The timestamp value currently resolves to whole seconds only; fractional seconds are not supported.
- The *`node_type`*, or type of node or application which is performing the logging. In the cluster log, this is always `[MgmSrvr]`; in the data node log, it is always `[ndbd]`. `[NdbApi]` and other values are possible in logs generated by NDB API applications and tools.
- The *`level`* of the event, sometimes also referred to as its severity level or log level. See earlier in this section, as well as Section 23.6.3.1, “NDB Cluster Logging Management Commands”, for more information about severity levels.
- The ID of the node reporting the event (*`node_id`*).
- A *`message`* containing a description of the event. The most common types of events to appear in the log are connections and disconnections between different nodes in the cluster, and when checkpoints occur. In some cases, the description may contain status or other information.

A sample from an actual cluster log is shown here:

```
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 5 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 5 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 6 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 6 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: President restarts arbitration thread [state=1]
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 7 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 7 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 8 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 8 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 9 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 9 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 50 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 50 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Start phase 101 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Start phase 101 completed (system restart)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Started (mysql-8.0.34 ndb-8.0.34)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Started (mysql-8.0.34 ndb-8.0.34)
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 5: Node 50: API mysql-8.0.34 ndb-8.0.34
2021-06-10 10:01:07 [MgmtSrvr] INFO     -- Node 6: Node 50: API mysql-8.0.34 ndb-8.0.34
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 6: Prepare arbitrator node 50 [ticket=75fd00010fa8b608]
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 5: Started arbitrator node 50 [ticket=75fd00010fa8b608]
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 6: Communication to Node 100 opened
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 6: Communication to Node 101 opened
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 5: Communication to Node 100 opened
2021-06-10 10:01:08 [MgmtSrvr] INFO     -- Node 5: Communication to Node 101 opened
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Alloc node id 100 succeeded
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Nodeid 100 allocated for API at 127.0.0.1
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Node 100: mysqld --server-id=1
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Node 5: Node 100 Connected
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Node 6: Node 100 Connected
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Node 5: Node 100: API mysql-8.0.34 ndb-8.0.34
2021-06-10 10:01:36 [MgmtSrvr] INFO     -- Node 6: Node 100: API mysql-8.0.34 ndb-8.0.34
```

For additional information, see Section 23.6.3.2, “NDB Cluster Log Events”.

### 23.6.3.1 NDB Cluster Logging Management Commands

`ndb_mgm` supports a number of management commands related to the cluster log and node logs. In the listing that follows, *`node_id`* denotes either a storage node ID or the keyword `ALL`, which indicates that the command should be applied to all of the cluster's data nodes.

- `CLUSTERLOG ON`

  Turns the cluster log on.

- `CLUSTERLOG OFF`

  Turns the cluster log off.

- `CLUSTERLOG INFO`

  Provides information about cluster log settings.

- *`node_id`* `CLUSTERLOG` *`category=threshold`*

  Logs *`category`* events with priority less than or equal to *`threshold`* in the cluster log.

- `CLUSTERLOG FILTER` *`severity_level`*

  Toggles cluster logging of events of the specified *`severity_level`*.

The following table describes the default setting (for all data nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, it is reported in the cluster log.

**Note**

Events are reported per data node, and that the threshold can be set to different values on different nodes.

**Table 23.54 Cluster log categories, with default threshold setting**

| Category | Default threshold (All data nodes) |
|---|---|
| `STARTUP` | **7** |
| `SHUTDOWN` | **7** |
| `STATISTICS` | **7** |
| `CHECKPOINT` | **7** |
| `NODERESTART` | **7** |
| `CONNECTION` | **7** |
| `ERROR` | **15** |
| `INFO` | **7** |

The `STATISTICS` category can provide a great deal of useful data. See Section 23.6.3.3, “Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client”, for more information.

Thresholds are used to filter events within each category. For example, a `STARTUP` event with a priority of 3 is not logged unless the threshold for `STARTUP` is set to 3 or higher. Only events with priority 3 or lower are sent if the threshold is 3.

The following table shows the event severity levels.

**Note**

These correspond to Unix `syslog` levels, except for `LOG_EMERG` and `LOG_NOTICE`, which are not used or mapped.

**Table 23.55 Event severity levels**

| Severity Level Value | Severity | Description |
|---|---|---|
| 1 | ALERT | A condition that should be corrected immediately, such as a corrupted system database |
| 2 | CRITICAL | Critical conditions, such as device errors or insufficient resources |
| 3 | ERROR | Conditions that should be corrected, such as configuration errors |
| 4 | WARNING | Conditions that are not errors, but that might require special handling |
| 5 | INFO | Informational messages |
| 6 | DEBUG | Debugging messages used for NDBCLUSTER development |

Event severity levels can be turned on or off (using CLUSTERLOG FILTER—see above). If a severity level is turned on, then all events with a priority less than or equal to the category thresholds are logged. If the severity level is turned off then no events belonging to that severity level are logged.

**Important**

Cluster log levels are set on a per ndb_mgmd, per subscriber basis. This means that, in an NDB Cluster with multiple management servers, using a CLUSTERLOG command in an instance of ndb_mgm connected to one management server affects only logs generated by that management server but not by any of the others. This also means that, should one of the management servers be restarted, only logs generated by that management server are affected by the resetting of log levels caused by the restart.

### 23.6.3.2 NDB Cluster Log Events

An event report reported in the event logs has the following format:

```
datetime [string] severity -- message
```

For example:

```
09:19:30 2005-07-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

This section discusses all reportable events, ordered by category and severity level within each category.

In the event descriptions, GCP and LCP mean “Global Checkpoint” and “Local Checkpoint”, respectively.

#### CONNECTION Events

These events are associated with connections between Cluster nodes.

**Table 23.56 Events associated with connections between cluster nodes**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| Connected | 8 | INFO | Data nodes connected |
| Disconnected | 8 | ALERT | Data nodes disconnected |

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| CommunicationClosed | 8 | INFO | SQL node or data node connection closed |
| CommunicationOpened | 8 | INFO | SQL node or data node connection open |
| ConnectedApiVersion | 8 | INFO | Connection using API version |

### CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

**Table 23.57 Events associated with checkpoints**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| GlobalCheckpointStarted | 9 | INFO | Start of GCP: REDO log is written to disk |
| GlobalCheckpointCompleted | 10 | INFO | GCP finished |
| LocalCheckpointStarted | 7 | INFO | Start of LCP: data written to disk |
| LocalCheckpointCompleted | 7 | INFO | LCP completed normally |
| LCPStoppedInCalcKeepGci | 0 | ALERT | LCP stopped |
| LCPFragmentCompleted | 11 | INFO | LCP on a fragment has been completed |
| UndoLogBlocked | 7 | INFO | UNDO logging blocked; buffer near overflow |
| RedoStatus | 7 | INFO | Redo status |

### STARTUP Events

The following events are generated in response to the startup of a node or of the cluster and of its success or failure. They also provide information relating to the progress of the startup process, including information concerning logging activities.

**Table 23.58 Events relating to the startup of a node or cluster**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| NDBStartStarted | 1 | INFO | Data node start phases initiated (all nodes starting) |
| NDBStartCompleted | 1 | INFO | Start phases completed, all data nodes |
| STTORRYRecieved | 15 | INFO | Blocks received after completion of restart |
| StartPhaseCompleted | 4 | INFO | Data node start phase *X* completed |
| CM_REGCONF | 3 | INFO | Node has been successfully included into the cluster; shows the node, managing node, and dynamic ID |
| CM_REGREF | 8 | INFO | Node has been refused for inclusion in the |

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| | | | cluster; cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem |
| FIND_NEIGHBOURS | 8 | INFO | Shows neighboring data nodes |
| NDBStopStarted | 1 | INFO | Data node shutdown initiated |
| NDBStopCompleted | 1 | INFO | Data node shutdown complete |
| NDBStopForced | 1 | ALERT | Forced shutdown of data node |
| NDBStopAborted | 1 | INFO | Unable to shut down data node normally |
| StartREDOLog | 4 | INFO | New redo log started; GCI keep *X*, newest restorable GCI *Y* |
| StartLog | 10 | INFO | New log started; log part *X*, start MB *Y*, stop MB *Z* |
| UNDORecordsExecuted | 15 | INFO | Undo records executed |
| StartReport | 4 | INFO | Report started |
| LogFileInitStatus | 7 | INFO | Log file initialization status |
| LogFileInitCompStatus | 7 | INFO | Log file completion status |
| StartReadLCP | 10 | INFO | Start read for local checkpoint |
| ReadLCPComplete | 10 | INFO | Read for local checkpoint completed |
| RunRedo | 8 | INFO | Running the redo log |
| RebuildIndex | 10 | INFO | Rebuilding indexes |

### NODERESTART Events

The following events are generated when restarting a node and relate to the success or failure of the node restart process.

**Table 23.59 Events relating to restarting a node**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| NR_CopyDict | 7 | INFO | Completed copying of dictionary information |
| NR_CopyDistr | 7 | INFO | Completed copying distribution information |
| NR_CopyFragsStarted | 7 | INFO | Starting to copy fragments |
| NR_CopyFragDone | 10 | INFO | Completed copying a fragment |

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| NR_CopyFragsCompleted | 7 | INFO | Completed copying all fragments |
| NodeFailCompleted | 8 | ALERT | Node failure phase completed |
| NODE_FAILREP | 8 | ALERT | Reports that a node has failed |
| ArbitState | 6 | INFO | Report whether an arbitrator is found or not; there are seven different possible outcomes when seeking an arbitrator, listed here:<br>• Management server restarts arbitration thread [state=*X*]<br>• Prepare arbitrator node *X* [ticket=*Y*]<br>• Receive arbitrator node *X* [ticket=*Y*]<br>• Started arbitrator node *X* [ticket=*Y*]<br>• Lost arbitrator node *X* - process failure [state=*Y*]<br>• Lost arbitrator node *X* - process exit [state=*Y*]<br>• Lost arbitrator node *X* <error msg> [state=*Y*] |
| ArbitResult | 2 | ALERT | Report arbitrator results; there are eight different possible results for arbitration attempts, listed here:<br>• Arbitration check failed: less than 1/2 nodes left<br>• Arbitration check succeeded: node group majority<br>• Arbitration check failed: missing node group<br>• Network partitioning: arbitration required |

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| | | | • Arbitration succeeded: affirmative response from node *X*<br>• Arbitration failed: negative response from node *X*<br>• Network partitioning: no arbitrator available<br>• Network partitioning: no arbitrator configured |
| GCP_TakeoverStarted | 7 | INFO | GCP takeover started |
| GCP_TakeoverCompleted | 7 | INFO | GCP takeover complete |
| LCP_TakeoverStarted | 7 | INFO | LCP takeover started |
| LCP_TakeoverCompleted | 7 | INFO | LCP takeover complete (state = *X*) |
| ConnectCheckStarted | 6 | INFO | Connection check started |
| ConnectCheckCompleted | 6 | INFO | Connection check completed |
| NodeFailRejected | 6 | ALERT | Node failure phase failed |

### STATISTICS Events

The following events are of a statistical nature. They provide information such as numbers of transactions and other operations, amount of data sent or received by individual nodes, and memory usage.

**Table 23.60 Events of a statistical nature**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| TransReportCounters | 8 | INFO | Report transaction statistics, including numbers of transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts |
| OperationReportCounters | 8 | INFO | Number of operations |
| TableCreated | 7 | INFO | Report number of tables created |
| JobStatistic | 9 | INFO | Mean internal job scheduling statistics |
| ThreadConfigLoop | 9 | INFO | Number of thread configuration loops |
| SendBytesStatistic | 9 | INFO | Mean number of bytes sent to node *X* |
| ReceiveBytesStatistic | 9 | INFO | Mean number of bytes received from node *X* |

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| MemoryUsage | 5 | INFO | Data and index memory usage (80%, 90%, and 100%) |
| MTSignalStatistics | 9 | INFO | Multithreaded signals |

### SCHEMA Events

These events relate to NDB Cluster schema operations.

**Table 23.61 Events relating to NDB Cluster schema operations**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| CreateSchemaObject | 8 | INFO | Schema objected created |
| AlterSchemaObject | 8 | INFO | Schema object updated |
| DropSchemaObject | 8 | INFO | Schema object dropped |

### ERROR Events

These events relate to Cluster errors and warnings. The presence of one or more of these generally indicates that a major malfunction or failure has occurred.

**Table 23.62 Events relating to cluster errors and warnings**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| TransporterError | 2 | ERROR | Transporter error |
| TransporterWarning | 8 | WARNING | Transporter warning |
| MissedHeartbeat | 8 | WARNING | Node *X* missed heartbeat number *Y* |
| DeadDueToHeartbeat | 8 | ALERT | Node *X* declared “dead” due to missed heartbeat |
| WarningEvent | 2 | WARNING | General warning event |
| SubscriptionStatus | 4 | WARNING | Change in subscription status |

### INFO Events

These events provide general information about the state of the cluster and activities associated with Cluster maintenance, such as logging and heartbeat transmission.

**Table 23.63 Information events**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| SentHeartbeat | 12 | INFO | Sent heartbeat |
| CreateLogBytes | 11 | INFO | Create log: Log part, log file, size in MB |
| InfoEvent | 2 | INFO | General informational event |
| EventBufferStatus | 7 | INFO | Event buffer status |
| EventBufferStatus2 | 7 | INFO | Improved event buffer status information |

**Note**

`SentHeartbeat` events are available only if NDB Cluster was compiled with `VM_TRACE` enabled.

### SINGLEUSER Events

These events are associated with entering and exiting single user mode.

**Table 23.64 Events relating to single user mode**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| `SingleUser` | 7 | `INFO` | Entering or exiting single user mode |

### BACKUP Events

These events provide information about backups being created or restored.

**Table 23.65 Backup events**

| Event | Priority | Severity Level | Description |
|---|---|---|---|
| `BackupStarted` | 7 | `INFO` | Backup started |
| `BackupStatus` | 7 | `INFO` | Backup status |
| `BackupCompleted` | 7 | `INFO` | Backup completed |
| `BackupFailedToStart` | 7 | `ALERT` | Backup failed to start |
| `BackupAborted` | 7 | `ALERT` | Backup aborted by user |
| `RestoreStarted` | 7 | `INFO` | Started restoring from backup |
| `RestoreMetaData` | 7 | `INFO` | Restoring metadata |
| `RestoreData` | 7 | `INFO` | Restoring data |
| `RestoreLog` | 7 | `INFO` | Restoring log files |
| `RestoreCompleted` | 7 | `INFO` | Completed restoring from backup |
| `SavedEvent` | 7 | `INFO` | Event saved |

## 23.6.3.3 Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client

The `NDB` management client's `CLUSTERLOG STATISTICS` command can provide a number of useful statistics in its output. Counters providing information about the state of the cluster are updated at 5-second reporting intervals by the transaction coordinator (TC) and the local query handler (LQH), and written to the cluster log.

**Transaction coordinator statistics.** Each transaction has one transaction coordinator, which is chosen by one of the following methods:

- In a round-robin fashion
- By communication proximity
- By supplying a data placement hint when the transaction is started

**Note**

You can determine which TC selection method is used for transactions started from a given SQL node using the `ndb_optimized_node_selection` system variable.

All operations within the same transaction use the same transaction coordinator, which reports the following statistics:

- **Trans count.** This is the number transactions started in the last interval using this TC as the transaction coordinator. Any of these transactions may have committed, have been aborted, or remain uncommitted at the end of the reporting interval.

  **Note**

  Transactions do not migrate between TCs.

- **Commit count.** This is the number of transactions using this TC as the transaction coordinator that were committed in the last reporting interval. Because some transactions committed in this reporting interval may have started in a previous reporting interval, it is possible for `Commit count` to be greater than `Trans count`.

- **Read count.** This is the number of primary key read operations using this TC as the transaction coordinator that were started in the last reporting interval, including simple reads. This count also includes reads performed as part of unique index operations. A unique index read operation generates 2 primary key read operations—1 for the hidden unique index table, and 1 for the table on which the read takes place.

- **Simple read count.** This is the number of simple read operations using this TC as the transaction coordinator that were started in the last reporting interval.

- **Write count.** This is the number of primary key write operations using this TC as the transaction coordinator that were started in the last reporting interval. This includes all inserts, updates, writes and deletes, as well as writes performed as part of unique index operations.

  **Note**

  A unique index update operation can generate multiple PK read and write operations on the index table and on the base table.

- **AttrInfoCount.** This is the number of 32-bit data words received in the last reporting interval for primary key operations using this TC as the transaction coordinator. For reads, this is proportional to the number of columns requested. For inserts and updates, this is proportional to the number of columns written, and the size of their data. For delete operations, this is usually zero.

  Unique index operations generate multiple PK operations and so increase this count. However, data words sent to describe the PK operation itself, and the key information sent, are *not* counted here. Attribute information sent to describe columns to read for scans, or to describe ScanFilters, is also not counted in `AttrInfoCount`.

- **Concurrent Operations.** This is the number of primary key or scan operations using this TC as the transaction coordinator that were started during the last reporting interval but that were not completed. Operations increment this counter when they are started and decrement it when they are completed; this occurs after the transaction commits. Dirty reads and writes—as well as failed operations—decrement this counter.

  The maximum value that `Concurrent Operations` can have is the maximum number of operations that a TC block can support; currently, this is `(2 * MaxNoOfConcurrentOperations) + 16 + MaxNoOfConcurrentTransactions`. (For more information about these configuration parameters, see the *Transaction Parameters* section of Section 23.4.3.6, “Defining NDB Cluster Data Nodes”.)

- **Abort count.** This is the number of transactions using this TC as the transaction coordinator that were aborted during the last reporting interval. Because some transactions that were aborted in the last reporting interval may have started in a previous reporting interval, `Abort count` can sometimes be greater than `Trans count`.

- **Scans.** This is the number of table scans using this TC as the transaction coordinator that were started during the last reporting interval. This does not include range scans (that is, ordered index scans).
- **Range scans.** This is the number of ordered index scans using this TC as the transaction coordinator that were started in the last reporting interval.
- **Local reads.** This is the number of primary-key read operations performed using a transaction coordinator on a node that also holds the primary fragment replica of the record. This count can also be obtained from the `LOCAL_READS` counter in the `ndbinfo.counters` table.
- **Local writes.** This contains the number of primary-key read operations that were performed using a transaction coordinator on a node that also holds the primary fragment replica of the record. This count can also be obtained from the `LOCAL_WRITES` counter in the `ndbinfo.counters` table.

**Local query handler statistics (Operations).** There is 1 cluster event per local query handler block (that is, 1 per data node process). Operations are recorded in the LQH where the data they are operating on resides.

**Note**

A single transaction may operate on data stored in multiple LQH blocks.

The `Operations` statistic provides the number of local operations performed by this LQH block in the last reporting interval, and includes all types of read and write operations (insert, update, write, and delete operations). This also includes operations used to replicate writes. For example, in a cluster with two fragment replicas, the write to the primary fragment replica is recorded in the primary LQH, and the write to the backup is recorded in the backup LQH. Unique key operations may result in multiple local operations; however, this does *not* include local operations generated as a result of a table scan or ordered index scan, which are not counted.

**Process scheduler statistics.** In addition to the statistics reported by the transaction coordinator and local query handler, each `ndbd` process has a scheduler which also provides useful metrics relating to the performance of an NDB Cluster. This scheduler runs in an infinite loop; during each loop the scheduler performs the following tasks:

1. Read any incoming messages from sockets into a job buffer.
2. Check whether there are any timed messages to be executed; if so, put these into the job buffer as well.
3. Execute (in a loop) any messages in the job buffer.
4. Send any distributed messages that were generated by executing the messages in the job buffer.
5. Wait for any new incoming messages.

Process scheduler statistics include the following:

- **Mean Loop Counter.** This is the number of loops executed in the third step from the preceding list. This statistic increases in size as the utilization of the TCP/IP buffer improves. You can use this to monitor changes in performance as you add new data node processes.
- **Mean send size and Mean receive size.** These statistics enable you to gauge the efficiency of, respectively writes and reads between nodes. The values are given in bytes. Higher values mean a lower cost per byte sent or received; the maximum value is 64K.

To cause all cluster log statistics to be logged, you can use the following command in the `NDB` management client:

```
ndb_mgm> ALL CLUSTERLOG STATISTICS=15
```

**Note**

Setting the threshold for `STATISTICS` to 15 causes the cluster log to become very verbose, and to grow quite rapidly in size, in direct proportion to the number of cluster nodes and the amount of activity in the NDB Cluster.

For more information about NDB Cluster management client commands relating to logging and reporting, see Section 23.6.3.1, “NDB Cluster Logging Management Commands”.

## 23.6.4 Summary of NDB Cluster Start Phases

This section provides a simplified outline of the steps involved when NDB Cluster data nodes are started. More complete information can be found in NDB Cluster Start Phases, in the *NDB Internals Guide*.

These phases are the same as those reported in the output from the *`node_id`* `STATUS` command in the management client (see Section 23.6.1, “Commands in the NDB Cluster Management Client”). These start phases are also reported in the `start_phase` column of the `ndbinfo.nodes` table.

**Start types.** There are several different startup types and modes, as shown in the following list:

- **Initial start.** The cluster starts with a clean file system on all data nodes. This occurs either when the cluster started for the very first time, or when all data nodes are restarted using the `--initial` option.

  **Note**

  Disk Data files are not removed when restarting a node using `--initial`.

- **System restart.** The cluster starts and reads data stored in the data nodes. This occurs when the cluster has been shut down after having been in use, when it is desired for the cluster to resume operations from the point where it left off.
- **Node restart.** This is the online restart of a cluster node while the cluster itself is running.
- **Initial node restart.** This is the same as a node restart, except that the node is reinitialized and started with a clean file system.

**Setup and initialization (phase -1).** Prior to startup, each data node (`ndbd` process) must be initialized. Initialization consists of the following steps:

1. Obtain a node ID
2. Fetch configuration data
3. Allocate ports to be used for inter-node communications
4. Allocate memory according to settings obtained from the configuration file

When a data node or SQL node first connects to the management node, it reserves a cluster node ID. To make sure that no other node allocates the same node ID, this ID is retained until the node has managed to connect to the cluster and at least one `ndbd` reports that this node is connected. This retention of the node ID is guarded by the connection between the node in question and `ndb_mgmd`.

After each data node has been initialized, the cluster startup process can proceed. The stages which the cluster goes through during this process are listed here:

- **Phase 0.** The `NDBFS` and `NDBCNTR` blocks start. Data node file systems are cleared on those data nodes that were started with `--initial` option.
- **Phase 1.** In this stage, all remaining `NDB` kernel blocks are started. NDB Cluster connections are set up, inter-block communications are established, and heartbeats are started. In the case of a node restart, API node connections are also checked.

**Note**

When one or more nodes hang in Phase 1 while the remaining node or nodes hang in Phase 2, this often indicates network problems. One possible cause of such issues is one or more cluster hosts having multiple network interfaces. Another common source of problems causing this condition is the blocking of TCP/IP ports needed for communications between cluster nodes. In the latter case, this is often due to a misconfigured firewall.

- **Phase 2.** The `NDBCNTR` kernel block checks the states of all existing nodes. The master node is chosen, and the cluster schema file is initialized.
- **Phase 3.** The `DBLQH` and `DBTC` kernel blocks set up communications between them. The startup type is determined; if this is a restart, the `DBDIH` block obtains permission to perform the restart.
- **Phase 4.** For an initial start or initial node restart, the redo log files are created. The number of these files is equal to `NoOfFragmentLogFiles`.

  For a system restart:

  - Read schema or schemas.
  - Read data from the local checkpoint.
  - Apply all redo information until the latest restorable global checkpoint has been reached.

  For a node restart, find the tail of the redo log.
- **Phase 5.** Most of the database-related portion of a data node start is performed during this phase. For an initial start or system restart, a local checkpoint is executed, followed by a global checkpoint. Periodic checks of memory usage begin during this phase, and any required node takeovers are performed.
- **Phase 6.** In this phase, node groups are defined and set up.
- **Phase 7.** The arbitrator node is selected and begins to function. The next backup ID is set, as is the backup disk write speed. Nodes reaching this start phase are marked as `Started`. It is now possible for API nodes (including SQL nodes) to connect to the cluster.
- **Phase 8.** If this is a system restart, all indexes are rebuilt (by `DBDIH`).
- **Phase 9.** The node internal startup variables are reset.
- **Phase 100 (OBSOLETE).** Formerly, it was at this point during a node restart or initial node restart that API nodes could connect to the node and begin to receive events. Currently, this phase is empty.
- **Phase 101.** At this point in a node restart or initial node restart, event delivery is handed over to the node joining the cluster. The newly-joined node takes over responsibility for delivering its primary data to subscribers. This phase is also referred to as *`SUMA` handover phase*.

After this process is completed for an initial start or system restart, transaction handling is enabled. For a node restart or initial node restart, completion of the startup process means that the node may now act as a transaction coordinator.

## 23.6.5 Performing a Rolling Restart of an NDB Cluster

This section discusses how to perform a *rolling restart* of an NDB Cluster installation, so called because it involves stopping and starting (or restarting) each node in turn, so that the cluster itself remains operational. This is often done as part of a *rolling upgrade* or *rolling downgrade*, where high availability of the cluster is mandatory and no downtime of the cluster as a whole is permissible. Where we refer to upgrades, the information provided here also generally applies to downgrades as well.

There are a number of reasons why a rolling restart might be desirable. These are described in the next few paragraphs.

**Configuration change.**
To make a change in the cluster's configuration, such as adding an SQL node to the cluster, or setting a configuration parameter to a new value.

**NDB Cluster software upgrade or downgrade.** To upgrade the cluster to a newer version of the NDB Cluster software (or to downgrade it to an older version). This is usually referred to as a “rolling upgrade” (or “rolling downgrade”, when reverting to an older version of NDB Cluster).

**Change on node host.** To make changes in the hardware or operating system on which one or more NDB Cluster node processes are running.

**System reset (cluster reset).**
To reset the cluster because it has reached an undesirable state. In such cases it is often desirable to reload the data and metadata of one or more data nodes. This can be done in any of three ways:

- Start each data node process (`ndbd` or possibly `ndbmtd`) with the `--initial` option, which forces the data node to clear its file system and to reload all NDB Cluster data and metadata from the other data nodes.

  Beginning with NDB 8.0.21, this also forces the removal of all Disk Data objects and files associated with those objects.

- Create a backup using the `ndb_mgm` client `START BACKUP` command prior to performing the restart. Following the upgrade, restore the node or nodes using `ndb_restore`.

  See Section 23.6.8, “Online Backup of NDB Cluster”, and Section 23.5.23, “ndb_restore — Restore an NDB Cluster Backup”, for more information.

- Use `mysqldump` to create a backup prior to the upgrade; afterward, restore the dump using `LOAD DATA`.

**Resource Recovery.**
To free memory previously allocated to a table by successive `INSERT` and `DELETE` operations, for re-use by other NDB Cluster tables.

The process for performing a rolling restart may be generalized as follows:

1. Stop all cluster management nodes (`ndb_mgmd` processes), reconfigure them, then restart them. (See Rolling restarts with multiple management servers.)

2. Stop, reconfigure, then restart each cluster data node (`ndbd` process) in turn.

   Some node configuration parameters can be updated by issuing `RESTART` for each of the data nodes in the `ndb_mgm` client following the previous step. Other parameters require that the data node be stopped completely using the management client `STOP` command, then started again from a system shell by invoking the `ndbd` or `ndbmtd` executable as appropriate. (A shell command such as `kill` can also be used on most Unix systems to stop a data node process, but the `STOP` command is preferred and usually simpler.)

   **Note**

   On Windows, you can also use `SC STOP` and `SC START` commands, `NET STOP` and `NET START` commands, or the Windows Service Manager to stop and start nodes which have been installed as Windows services (see Section 23.3.2.4, “Installing NDB Cluster Processes as Windows Services”).

   The type of restart required is indicated in the documentation for each node configuration parameter. See Section 23.4.3, “NDB Cluster Configuration Files”.

3. Stop, reconfigure, then restart each cluster SQL node (`mysqld` process) in turn.

NDB Cluster supports a somewhat flexible order for upgrading nodes. When upgrading an NDB Cluster, you may upgrade API nodes (including SQL nodes) before upgrading the management nodes, data nodes, or both. In other words, you are permitted to upgrade the API and SQL nodes in any order. This is subject to the following provisions:

- This functionality is intended for use as part of an online upgrade only. A mix of node binaries from different NDB Cluster releases is neither intended nor supported for continuous, long-term use in a production setting.
- You must upgrade all nodes of the same type (management, data, or API node) before upgrading any nodes of a different type. This remains true regardless of the order in which the nodes are upgraded.
- You must upgrade all management nodes before upgrading any data nodes. This remains true regardless of the order in which you upgrade the cluster's API and SQL nodes.
- Features specific to the “new” version must not be used until all management nodes and data nodes have been upgraded.

  This also applies to any MySQL Server version change that may apply, in addition to the NDB engine version change, so do not forget to take this into account when planning the upgrade. (This is true for online upgrades of NDB Cluster in general.)

It is not possible for any API node to perform schema operations (such as data definition statements) during a node restart. Due in part to this limitation, schema operations are also not supported during an online upgrade or downgrade. In addition, it is not possible to perform native backups while an upgrade or downgrade is ongoing.

**Rolling restarts with multiple management servers.** When performing a rolling restart of an NDB Cluster with multiple management nodes, you should keep in mind that `ndb_mgmd` checks to see if any other management node is running, and, if so, tries to use that node's configuration data. To keep this from occurring, and to force `ndb_mgmd` to re-read its configuration file, perform the following steps:

1. Stop all NDB Cluster `ndb_mgmd` processes.
2. Update all `config.ini` files.
3. Start a single `ndb_mgmd` with `--reload`, `--initial`, or both options as desired.
4. If you started the first `ndb_mgmd` with the `--initial` option, you must also start any remaining `ndb_mgmd` processes using `--initial`.

   Regardless of any other options used when starting the first `ndb_mgmd`, you should not start any remaining `ndb_mgmd` processes after the first one using `--reload`.
5. Complete the rolling restarts of the data nodes and API nodes as normal.

When performing a rolling restart to update the cluster's configuration, you can use the `config_generation` column of the `ndbinfo.nodes` table to keep track of which data nodes have been successfully restarted with the new configuration. See Section 23.6.16.47, “The ndbinfo nodes Table”.

## 23.6.6 NDB Cluster Single User Mode

*Single user mode* enables the database administrator to restrict access to the database system to a single API node, such as a MySQL server (SQL node) or an instance of `ndb_restore`. When entering single user mode, connections to all other API nodes are closed gracefully and all running transactions are aborted. No new transactions are permitted to start.

Once the cluster has entered single user mode, only the designated API node is granted access to the database.

You can use the `ALL STATUS` command in the `ndb_mgm` client to see when the cluster has entered single user mode. You can also check the `status` column of the `ndbinfo.nodes` table (see Section 23.6.16.47, “The ndbinfo nodes Table”, for more information).

Example:

```
ndb_mgm> ENTER SINGLE USER MODE 5
```

After this command has executed and the cluster has entered single user mode, the API node whose node ID is `5` becomes the cluster's only permitted user.

The node specified in the preceding command must be an API node; attempting to specify any other type of node is rejected.

**Note**

When the preceding command is invoked, all transactions running on the designated node are aborted, the connection is closed, and the server must be restarted.

The command `EXIT SINGLE USER MODE` changes the state of the cluster's data nodes from single user mode to normal mode. API nodes—such as MySQL Servers—waiting for a connection (that is, waiting for the cluster to become ready and available), are again permitted to connect. The API node denoted as the single-user node continues to run (if still connected) during and after the state change.

Example:

```
ndb_mgm> EXIT SINGLE USER MODE
```

There are two recommended ways to handle a node failure when running in single user mode:

- Method 1:
  1. Finish all single user mode transactions
  2. Issue the `EXIT SINGLE USER MODE` command
  3. Restart the cluster's data nodes
- Method 2:

  Restart storage nodes prior to entering single user mode.

## 23.6.7 Adding NDB Cluster Data Nodes Online

This section describes how to add NDB Cluster data nodes “online”—that is, without needing to shut down the cluster completely and restart it as part of the process.

**Important**

Currently, you must add new data nodes to an NDB Cluster as part of a new node group. In addition, it is not possible to change the number of fragment replicas (or the number of nodes per node group) online.

### 23.6.7.1 Adding NDB Cluster Data Nodes Online: General Issues

This section provides general information about the behavior of and current limitations in adding NDB Cluster nodes online.

**Redistribution of Data.** The ability to add new nodes online includes a means to reorganize `NDBCLUSTER` table data and indexes so that they are distributed across all data nodes, including the new ones, by means of the `ALTER TABLE ... REORGANIZE PARTITION` statement. Table reorganization of both in-memory and Disk Data tables is supported. This redistribution does not currently include unique indexes (only ordered indexes are redistributed).

The redistribution for `NDBCLUSTER` tables already existing before the new data nodes were added is not automatic, but can be accomplished using simple SQL statements in `mysql` or another MySQL client application. However, all data and indexes added to tables created after a new node group has been added are distributed automatically among all cluster data nodes, including those added as part of the new node group.

**Partial starts.** It is possible to add a new node group without all of the new data nodes being started. It is also possible to add a new node group to a degraded cluster—that is, a cluster that is only partially started, or where one or more data nodes are not running. In the latter case, the cluster must have enough nodes running to be viable before the new node group can be added.

**Effects on ongoing operations.** Normal DML operations using NDB Cluster data are not prevented by the creation or addition of a new node group, or by table reorganization. However, it is not possible to perform DDL concurrently with table reorganization—that is, no other DDL statements can be issued while an `ALTER TABLE ... REORGANIZE PARTITION` statement is executing. In addition, during the execution of `ALTER TABLE ... REORGANIZE PARTITION` (or the execution of any other DDL statement), it is not possible to restart cluster data nodes.

**Failure handling.** Failures of data nodes during node group creation and table reorganization are handled as shown in the following table:

**Table 23.66 Data node failure handling during node group creation and table reorganization**

| Failure during | Failure in “Old” data node | Failure in “New” data node | System Failure |
|---|---|---|---|
| Node group creation | • **If a node other than the master fails:** The creation of the node group is always rolled forward.<br>• **If the master fails:**<br>  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.<br>  • **If the internal commit point has not yet been reached.** The creation of the node group is rolled back | • **If a node other than the master fails:** The creation of the node group is always rolled forward.<br>• **If the master fails:**<br>  • **If the internal commit point has been reached:** The creation of the node group is rolled forward.<br>  • **If the internal commit point has not yet been reached.** The creation of the node group is rolled back | • **If the execution of CREATE NODEGROUP has reached the internal commit point:** When restarted, the cluster includes the new node group. Otherwise it without.<br>• **If the execution of CREATE NODEGROUP has not yet reached the internal commit point:** When restarted, the cluster does not include the new node group. |
| Table reorganization | • **If a node other than the master fails:** The table reorganization is always rolled forward.<br>• **If the master fails:**<br>  • **If the internal commit point has been reached:** The | • **If a node other than the master fails:** The table reorganization is always rolled forward.<br>• **If the master fails:**<br>  • **If the internal commit point has been reached:** The | • **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using the “new” data nodes. |

| Failure during | Failure in “Old” data node | Failure in “New” data node | System Failure |
|---|---|---|---|
| | table reorganization is rolled forward.<br>• **If the internal commit point has not yet been reached.** The table reorganization is rolled back. | table reorganization is rolled forward.<br>• **If the internal commit point has not yet been reached.** The table reorganization is rolled back. | • **If the execution of an ALTER TABLE ... REORGANIZE PARTITION statement has not yet reached the internal commit point:** When the cluster is restarted, the data and indexes belonging to *`table`* are distributed using only the “old” data nodes. |

**Dropping node groups.** The `ndb_mgm` client supports a `DROP NODEGROUP` command, but it is possible to drop a node group only when no data nodes in the node group contain any data. Since there is currently no way to “empty” a specific data node or node group, this command works only the following two cases:

1. After issuing `CREATE NODEGROUP` in the `ndb_mgm` client, but before issuing any `ALTER TABLE ... REORGANIZE PARTITION` statements in the `mysql` client.
2. After dropping all `NDBCLUSTER` tables using `DROP TABLE`.

   `TRUNCATE TABLE` does not work for this purpose because the data nodes continue to store the table definitions.

### 23.6.7.2 Adding NDB Cluster Data Nodes Online: Basic procedure

In this section, we list the basic steps required to add new data nodes to an NDB Cluster. This procedure applies whether you are using `ndbd` or `ndbmtd` binaries for the data node processes. For a more detailed example, see Section 23.6.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

Assuming that you already have a running NDB Cluster, adding data nodes online requires the following steps:

1. Edit the cluster configuration `config.ini` file, adding new `[ndbd]` sections corresponding to the nodes to be added. In the case where the cluster uses multiple management servers, these changes need to be made to all `config.ini` files used by the management servers.

   You must be careful that node IDs for any new data nodes added in the `config.ini` file do not overlap node IDs used by existing nodes. In the event that you have API nodes using dynamically allocated node IDs and these IDs match node IDs that you want to use for new data nodes, it is possible to force any such API nodes to “migrate”, as described later in this procedure.

2. Perform a rolling restart of all NDB Cluster management servers.

   **Important**

   All management servers must be restarted with the `--reload` or `--initial` option to force the reading of the new configuration.

3. Perform a rolling restart of all existing NDB Cluster data nodes. It is not necessary (or usually even desirable) to use `--initial` when restarting the existing data nodes.

   If you are using API nodes with dynamically allocated IDs matching any node IDs that you wish to assign to new data nodes, you must restart all API nodes (including SQL nodes) before restarting

any of the data nodes processes in this step. This causes any API nodes with node IDs that were previously not explicitly assigned to relinquish those node IDs and acquire new ones.

4. Perform a rolling restart of any SQL or API nodes connected to the NDB Cluster.

5. Start the new data nodes.

   The new data nodes may be started in any order. They can also be started concurrently, as long as they are started after the rolling restarts of all existing data nodes have been completed, and before proceeding to the next step.

6. Execute one or more `CREATE NODEGROUP` commands in the NDB Cluster management client to create the new node group or node groups to which the new data nodes belong.

7. Redistribute the cluster's data among all data nodes, including the new ones. Normally this is done by issuing an `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` statement in the `mysql` client for each `NDBCLUSTER` table.

   *Exception*: For tables created using the `MAX_ROWS` option, this statement does not work; instead, use `ALTER TABLE ... ALGORITHM=INPLACE MAX_ROWS=...` to reorganize such tables. You should also bear in mind that using `MAX_ROWS` to set the number of partitions in this fashion is deprecated, and you should use `PARTITION_BALANCE` instead; see Section 13.1.20.12, “Setting NDB Comment Options”, for more information.

   **Note**

   This needs to be done only for tables already existing at the time the new node group is added. Data in tables created after the new node group is added is distributed automatically; however, data added to any given table `tbl` that existed before the new nodes were added is not distributed using the new nodes until that table has been reorganized.

8. `ALTER TABLE ... REORGANIZE PARTITION ALGORITHM=INPLACE` reorganizes partitions but does not reclaim the space freed on the “old” nodes. You can do this by issuing, for each `NDBCLUSTER` table, an `OPTIMIZE TABLE` statement in the `mysql` client.

   This works for space used by variable-width columns of in-memory `NDB` tables. `OPTIMIZE TABLE` is not supported for fixed-width columns of in-memory tables; it is also not supported for Disk Data tables.

You can add all the nodes desired, then issue several `CREATE NODEGROUP` commands in succession to add the new node groups to the cluster.

### 23.6.7.3 Adding NDB Cluster Data Nodes Online: Detailed Example

In this section we provide a detailed example illustrating how to add new NDB Cluster data nodes online, starting with an NDB Cluster having 2 data nodes in a single node group and concluding with a cluster having 4 data nodes in 2 node groups.

**Starting configuration.** For purposes of illustration, we assume a minimal configuration, and that the cluster uses a `config.ini` file containing only the following information:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
```

```
HostName = 198.51.100.2

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

**Note**

We have left a gap in the sequence between data node IDs and other nodes. This make it easier later to assign node IDs that are not already in use to data nodes which are newly added.

We also assume that you have already started the cluster using the appropriate command line or `my.cnf` options, and that running `SHOW` in the management client produces output similar to what is shown here:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.34-ndb-8.0.34, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.34-ndb-8.0.34, Nodegroup: 0)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.34-ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.34-ndb-8.0.34)
id=21   @198.51.100.21  (8.0.34-ndb-8.0.34)
```

Finally, we assume that the cluster contains a single `NDBCLUSTER` table created as shown here:

```
USE n;

CREATE TABLE ips (
    id BIGINT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    country_code CHAR(2) NOT NULL,
    type CHAR(4) NOT NULL,
    ip_address VARCHAR(15) NOT NULL,
    addresses BIGINT UNSIGNED DEFAULT NULL,
    date BIGINT UNSIGNED DEFAULT NULL
)   ENGINE NDBCLUSTER;
```

The memory usage and related information shown later in this section was generated after inserting approximately 50000 rows into this table.

**Note**

In this example, we show the single-threaded `ndbd` being used for the data node processes. You can also apply this example, if you are using the multithreaded `ndbmtd` by substituting `ndbmtd` for `ndbd` wherever it appears in the steps that follow.

**Step 1: Update configuration file.** Open the cluster global configuration file in a text editor and add `[ndbd]` sections corresponding to the 2 new data nodes. (We give these data nodes IDs 3 and 4, and assume that they are to be run on host machines at addresses 198.51.100.3 and 198.51.100.4, respectively.) After you have added the new sections, the contents of the `config.ini` file should look like what is shown here, where the additions to the file are shown in bold type:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3

[ndbd]
Id = 4
HostName = 198.51.100.4

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

Once you have made the necessary changes, save the file.

**Step 2: Restart the management server.** Restarting the cluster management server requires that you issue separate commands to stop the management server and then to start it again, as follows:

1. Stop the management server using the management client `STOP` command, as shown here:

   ```
   ndb_mgm> 10 STOP
   Node 10 has shut down.
   Disconnecting to allow Management Server to shutdown

   $>
   ```

2. Because shutting down the management server causes the management client to terminate, you must start the management server from the system shell. For simplicity, we assume that `config.ini` is in the same directory as the management server binary, but in practice, you must supply the correct path to the configuration file. You must also supply the `--reload` or `--initial` option so that the management server reads the new configuration from the file rather than its configuration cache. If your shell's current directory is also the same as the directory where the management server binary is located, then you can invoke the management server as shown here:

   ```
   $> ndb_mgmd -f config.ini --reload
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- NDB Cluster Management Server. 8.0.34-ndb-8.0.34
   2008-12-08 17:29:23 [MgmSrvr] INFO     -- Reading cluster configuration from 'config.ini'
   ```

If you check the output of `SHOW` in the management client after restarting the `ndb_mgm` process, you should now see something like this:

```
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.34-ndb-8.0.34, Nodegroup: 0, *)
```

```
id=2    @198.51.100.2  (8.0.34-ndb-8.0.34, Nodegroup: 0)
id=3 (not connected, accepting connect from 198.51.100.3)
id=4 (not connected, accepting connect from 198.51.100.4)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.34-ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.34-ndb-8.0.34)
id=21   @198.51.100.21  (8.0.34-ndb-8.0.34)
```

**Step 3: Perform a rolling restart of the existing data nodes.** This step can be accomplished entirely within the cluster management client using the `RESTART` command, as shown here:

```
ndb_mgm> 1 RESTART
Node 1: Node shutdown initiated
Node 1: Node shutdown completed, restarting, no start.
Node 1 is being restarted

ndb_mgm> Node 1: Start initiated (version 8.0.34)
Node 1: Started (version 8.0.34)

ndb_mgm> 2 RESTART
Node 2: Node shutdown initiated
Node 2: Node shutdown completed, restarting, no start.
Node 2 is being restarted

ndb_mgm> Node 2: Start initiated (version 8.0.34)

ndb_mgm> Node 2: Started (version 8.0.34)
```

**Important**

After issuing each *`X`* `RESTART` command, wait until the management client reports `Node X: Started (version ...)` *before* proceeding any further.

You can verify that all existing data nodes were restarted using the updated configuration by checking the `ndbinfo.nodes` table in the `mysql` client.

**Step 4: Perform a rolling restart of all cluster API nodes.** Shut down and restart each MySQL server acting as an SQL node in the cluster using `mysqladmin shutdown` followed by `mysqld_safe` (or another startup script). This should be similar to what is shown here, where *`password`* is the MySQL `root` password for a given MySQL server instance:

```
$> mysqladmin -uroot -ppassword shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
/usr/local/mysql/var/tonfisk.pid ended
$> mysqld_safe --ndbcluster --ndb-connectstring=198.51.100.10 &
081208 20:20:06 mysqld_safe Logging to '/usr/local/mysql/var/tonfisk.err'.
081208 20:20:06 mysqld_safe Starting mysqld daemon with databases
from /usr/local/mysql/var
```

Of course, the exact input and output depend on how and where MySQL is installed on the system, as well as which options you choose to start it (and whether or not some or all of these options are specified in a `my.cnf` file).

**Step 5: Perform an initial start of the new data nodes.** From a system shell on each of the hosts for the new data nodes, start the data nodes as shown here, using the `--initial` option:

```
$> ndbd -c 198.51.100.10 --initial
```

**Note**

Unlike the case with restarting the existing data nodes, you can start the new data nodes concurrently; you do not need to wait for one to finish starting before starting the other.

*Wait until both of the new data nodes have started before proceeding with the next step*. Once the new data nodes have started, you can see in the output of the management client `SHOW` command that they do not yet belong to any node group (as indicated with bold type here):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.34-ndb-8.0.34, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.34-ndb-8.0.34, Nodegroup: 0)
id=3    @198.51.100.3  (8.0.34-ndb-8.0.34, no nodegroup)
id=4    @198.51.100.4  (8.0.34-ndb-8.0.34, no nodegroup)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.34-ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.34-ndb-8.0.34)
id=21   @198.51.100.21  (8.0.34-ndb-8.0.34)
```

**Step 6: Create a new node group.** You can do this by issuing a `CREATE NODEGROUP` command in the cluster management client. This command takes as its argument a comma-separated list of the node IDs of the data nodes to be included in the new node group, as shown here:

```
ndb_mgm> CREATE NODEGROUP 3,4
Nodegroup 1 created
```

By issuing `SHOW` again, you can verify that data nodes 3 and 4 have joined the new node group (again indicated in bold type):

```
ndb_mgm> SHOW
Connected to Management Server at: 198.51.100.10:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=1    @198.51.100.1  (8.0.34-ndb-8.0.34, Nodegroup: 0, *)
id=2    @198.51.100.2  (8.0.34-ndb-8.0.34, Nodegroup: 0)
id=3    @198.51.100.3  (8.0.34-ndb-8.0.34, Nodegroup: 1)
id=4    @198.51.100.4  (8.0.34-ndb-8.0.34, Nodegroup: 1)

[ndb_mgmd(MGM)] 1 node(s)
id=10   @198.51.100.10  (8.0.34-ndb-8.0.34)

[mysqld(API)]   2 node(s)
id=20   @198.51.100.20  (8.0.34-ndb-8.0.34)
id=21   @198.51.100.21  (8.0.34-ndb-8.0.34)
```

**Step 7: Redistribute cluster data.** When a node group is created, existing data and indexes are not automatically distributed to the new node group's data nodes, as you can see by issuing the appropriate `REPORT` command in the management client:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(177 32K pages of total 3200)
Node 1: Index usage is 0%(108 8K pages of total 12832)
Node 2: Data usage is 5%(177 32K pages of total 3200)
Node 2: Index usage is 0%(108 8K pages of total 12832)
Node 3: Data usage is 0%(0 32K pages of total 3200)
Node 3: Index usage is 0%(0 8K pages of total 12832)
Node 4: Data usage is 0%(0 32K pages of total 3200)
Node 4: Index usage is 0%(0 8K pages of total 12832)
```

By using `ndb_desc` with the `-p` option, which causes the output to include partitioning information, you can see that the table still uses only 2 partitions (in the `Per partition info` section of the output, shown here in bold text):

```
$> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 1
```

```
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 340
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 2
TableStatus: Retrieved
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           26086       26086         1572864             557056
1           26329       26329         1605632             557056

NDBT_ProgramExit: 0 - OK
```

You can cause the data to be redistributed among all of the data nodes by performing, for each `NDB` table, an `ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` statement in the `mysql` client.

**Important**

`ALTER TABLE ... ALGORITHM=INPLACE, REORGANIZE PARTITION` does not work on tables that were created with the `MAX_ROWS` option. Instead, use `ALTER TABLE ... ALGORITHM=INPLACE, MAX_ROWS=...` to reorganize such tables.

Keep in mind that using `MAX_ROWS` to set the number of partitions per table is deprecated, and you should use `PARTITION_BALANCE` instead; see Section 13.1.20.12, “Setting NDB Comment Options”, for more information.

After issuing the statement `ALTER TABLE ips ALGORITHM=INPLACE, REORGANIZE PARTITION`, you can see using `ndb_desc` that the data for this table is now stored using 4 partitions, as shown here (with the relevant portions of the output in bold type):

```
$> ndb_desc -c 198.51.100.10 -d n ips -p
-- ips --
Version: 16777217
Fragment type: 9
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 6
Number of primary keys: 1
Length of frm data: 341
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
FragmentCount: 4
TableStatus: Retrieved
```

```
-- Attributes --
id Bigint PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
country_code Char(2;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
type Char(4;latin1_swedish_ci) NOT NULL AT=FIXED ST=MEMORY
ip_address Varchar(15;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY
addresses Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

-- Per partition info --
Partition   Row count   Commit count  Frag fixed memory   Frag varsized memory
0           12981       52296         1572864             557056
1           13236       52515         1605632             557056
2           13105       13105         819200              294912
3           13093       13093         819200              294912

NDBT_ProgramExit: 0 - OK
```

**Note**

Normally, `ALTER TABLE table_name [ALGORITHM=INPLACE,] REORGANIZE PARTITION` is used with a list of partition identifiers and a set of partition definitions to create a new partitioning scheme for a table that has already been explicitly partitioned. Its use here to redistribute data onto a new NDB Cluster node group is an exception in this regard; when used in this way, no other keywords or identifiers follow `REORGANIZE PARTITION`.

For more information, see Section 13.1.9, “ALTER TABLE Statement”.

In addition, for each table, the `ALTER TABLE` statement should be followed by an `OPTIMIZE TABLE` to reclaim wasted space. You can obtain a list of all `NDBCLUSTER` tables using the following query against the Information Schema `TABLES` table:

```
SELECT TABLE_SCHEMA, TABLE_NAME
    FROM INFORMATION_SCHEMA.TABLES
    WHERE ENGINE = 'NDBCLUSTER';
```

**Note**

The `INFORMATION_SCHEMA.TABLES.ENGINE` value for an NDB Cluster table is always `NDBCLUSTER`, regardless of whether the `CREATE TABLE` statement used to create the table (or `ALTER TABLE` statement used to convert an existing table from a different storage engine) used `NDB` or `NDBCLUSTER` in its `ENGINE` option.

You can see after performing these statements in the output of `ALL REPORT MEMORY` that the data and indexes are now redistributed between all cluster data nodes, as shown here:

```
ndb_mgm> ALL REPORT MEMORY

Node 1: Data usage is 5%(176 32K pages of total 3200)
Node 1: Index usage is 0%(76 8K pages of total 12832)
Node 2: Data usage is 5%(176 32K pages of total 3200)
Node 2: Index usage is 0%(76 8K pages of total 12832)
Node 3: Data usage is 2%(80 32K pages of total 3200)
Node 3: Index usage is 0%(51 8K pages of total 12832)
Node 4: Data usage is 2%(80 32K pages of total 3200)
Node 4: Index usage is 0%(50 8K pages of total 12832)
```

**Note**

Since only one DDL operation on `NDBCLUSTER` tables can be executed at a time, you must wait for each `ALTER TABLE ... REORGANIZE PARTITION` statement to finish before issuing the next one.

It is not necessary to issue `ALTER TABLE ... REORGANIZE PARTITION` statements for `NDBCLUSTER` tables created *after* the new data nodes have been added; data added to such tables is distributed among all data nodes automatically. However, in `NDBCLUSTER` tables that existed *prior to* the addition of the new nodes, neither existing nor new data is distributed using the new nodes until these tables have been reorganized using `ALTER TABLE ... REORGANIZE PARTITION`.

**Alternative procedure, without rolling restart.** It is possible to avoid the need for a rolling restart by configuring the extra data nodes, but not starting them, when first starting the cluster. We assume, as before, that you wish to start with two data nodes—nodes 1 and 2—in one node group and later to expand the cluster to four data nodes, by adding a second node group consisting of nodes 3 and 4:

```
[ndbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster

[ndbd]
Id = 1
HostName = 198.51.100.1

[ndbd]
Id = 2
HostName = 198.51.100.2

[ndbd]
Id = 3
HostName = 198.51.100.3
Nodegroup = 65536

[ndbd]
Id = 4
HostName = 198.51.100.4
Nodegroup = 65536

[mgm]
HostName = 198.51.100.10
Id = 10

[api]
Id=20
HostName = 198.51.100.20

[api]
Id=21
HostName = 198.51.100.21
```

The data nodes to be brought online at a later time (nodes 3 and 4) can be configured with `NodeGroup = 65536`, in which case nodes 1 and 2 can each be started as shown here:

```
$> ndbd -c 198.51.100.10 --initial
```

The data nodes configured with `NodeGroup = 65536` are treated by the management server as though you had started nodes 1 and 2 using `--nowait-nodes=3,4` after waiting for a period of time determined by the setting for the `StartNoNodeGroupTimeout` data node configuration parameter. By default, this is 15 seconds (15000 milliseconds).

**Note**

`StartNoNodegroupTimeout` must be the same for all data nodes in the cluster; for this reason, you should always set it in the `[ndbd default]` section of the `config.ini` file, rather than for individual data nodes.

When you are ready to add the second node group, you need only perform the following additional steps:

1. Start data nodes 3 and 4, invoking the data node process once for each new node:

```
$> ndbd -c 198.51.100.10 --initial
```

2. Issue the appropriate `CREATE NODEGROUP` command in the management client:

```
ndb_mgm> CREATE NODEGROUP 3,4
```

3. In the `mysql` client, issue `ALTER TABLE ... REORGANIZE PARTITION` and `OPTIMIZE TABLE` statements for each existing `NDBCLUSTER` table. (As noted elsewhere in this section, existing NDB Cluster tables cannot use the new nodes for data distribution until this has been done.)

## 23.6.8 Online Backup of NDB Cluster

The next few sections describe how to prepare for and then to create an NDB Cluster backup using the functionality for this purpose found in the `ndb_mgm` management client. To distinguish this type of backup from a backup made using `mysqldump`, we sometimes refer to it as a “native” NDB Cluster backup. (For information about the creation of backups with `mysqldump`, see Section 4.5.4, “mysqldump — A Database Backup Program”.) Restoration of NDB Cluster backups is done using the `ndb_restore` utility provided with the NDB Cluster distribution; for information about `ndb_restore` and its use in restoring NDB Cluster backups, see Section 23.5.23, “ndb_restore — Restore an NDB Cluster Backup”.

NDB 8.0 makes it possible to create backups using multiple LDMs to achieve parallelism on the data nodes. See Section 23.6.8.5, “Taking an NDB Backup with Parallel Data Nodes”.

### 23.6.8.1 NDB Cluster Backup Concepts

A backup is a snapshot of the database at a given time. The backup consists of three main parts:

- **Metadata.** The names and definitions of all database tables
- **Table records.** The data actually stored in the database tables at the time that the backup was made
- **Transaction log.** A sequential record telling how and when data was stored in the database

Each of these parts is saved on all nodes participating in the backup. During backup, each node saves these three parts into three files on disk:

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.ctl`

  A control file containing control information and metadata. Each node saves the same table definitions (for all tables in the cluster) to its own version of this file.

- `BACKUP-`*`backup_id`*`-0.`*`node_id`*`.data`

  A data file containing the table records, which are saved on a per-fragment basis. That is, different nodes save different fragments during the backup. The file saved by each node starts with a header that states the tables to which the records belong. Following the list of records there is a footer containing a checksum for all records.

- `BACKUP-`*`backup_id`*`.`*`node_id`*`.log`

  A log file containing records of committed transactions. Only transactions on tables stored in the backup are stored in the log. Nodes involved in the backup save different records because different nodes host different database fragments.

In the listing just shown, *`backup_id`* stands for the backup identifier and *`node_id`* is the unique identifier for the node creating the file.

The location of the backup files is determined by the `BackupDataDir` parameter.

### 23.6.8.2 Using The NDB Cluster Management Client to Create a Backup

Before starting a backup, make sure that the cluster is properly configured for performing one. (See Section 23.6.8.3, “Configuration for NDB Cluster Backups”.)

The `START BACKUP` command is used to create a backup, and has the syntax shown here:

```
START BACKUP [backup_id]
    [encryption_option]
    [wait_option]
    [snapshot_option]

encryption_option:
ENCRYPT [PASSWORD=password]

password:
{'password_string' | "password_string"}

wait_option:
WAIT {STARTED | COMPLETED} | NOWAIT

snapshot_option:
SNAPSHOTSTART | SNAPSHOTEND
```

Successive backups are automatically identified sequentially, so the *`backup_id`*, an integer greater than or equal to 1, is optional; if it is omitted, the next available value is used. If an existing *`backup_id`* value is used, the backup fails with the error `Backup failed: file already exists`. If used, the *`backup_id`* must follow immediately after the `START BACKUP` keywords, before any other options are used.

In NDB 8.0.22 and later, `START BACKUP` supports the creation of encrypted backups using `ENCRYPT PASSWORD=`*`password`*. The *`password`* must meet all of the following requirements:

- Uses any of the printable ASCII characters except `!`, `'`, `"`, `$`, `%`, `\`, and `^`
- Is no more than 256 characters in length
- Is enclosed by single or double quotation marks

When `ENCRYPT PASSWORD='`*`password`*`'` is used, the backup data record and log files written by each data node are encrypted with a key derived from the user-provided *`password`* and a randomly-generated salt using a key derivation function (KDF) that employs the PBKDF2-SHA256 algorithm to generate a symmetric encryption key for that file. This function has the form shown here:

```
key = KDF(random_salt, password)
```

The key so generated is then used to encrypt the backup data using AES 256 CBC inline, and symmetric encryption is employed for encrypting the backup fileset (with the generated key).

**Note**

NDB Cluster *never* saves the user-furnished password or generated encryption key.

Starting with NDB 8.0.24, the `PASSWORD` option can be omitted from *`encryption_option`*. In this case, the management client prompts the user for a password.

It is possible using `PASSWORD` to set an empty password (`''` or `""`), but this is not recommended.

An encrypted backup can be decrypted using any of the following commands:

- `ndb_restore --decrypt --backup-password=`*`password`*
- `ndbxfrm --decrypt-password=`*`password input_file output_file`*
- `ndb_print_backup_file -P` *`password file_name`*

NDB 8.0.24 and later supports the additional commands listed here:

- `ndb_restore --decrypt --backup-password-from-stdin`
- `ndbxfrm --decrypt-password-from-stdin input_file output_file`
- `ndb_print_backup_file --backup-password=password file_name`
- `ndb_print_backup_file --backup-password-from-stdin file_name`
- `ndb_mgm --backup-password-from-stdin --execute "START BACKUP ..."`

See the descriptions of these programs for more information, such as additional options that may be required.

The *`wait_option`* can be used to determine when control is returned to the management client after a `START BACKUP` command is issued, as shown in the following list:

- If `NOWAIT` is specified, the management client displays a prompt immediately, as seen here:

```
ndb_mgm> START BACKUP NOWAIT
ndb_mgm>
```

  In this case, the management client can be used even while it prints progress information from the backup process.

- With `WAIT STARTED` the management client waits until the backup has started before returning control to the user, as shown here:

```
ndb_mgm> START BACKUP WAIT STARTED
Waiting for started, this may take several minutes
Node 2: Backup 3 started from node 1
ndb_mgm>
```

- **`WAIT COMPLETED`** causes the management client to wait until the backup process is complete before returning control to the user.

`WAIT COMPLETED` is the default.

A *`snapshot_option`* can be used to determine whether the backup matches the state of the cluster when `START BACKUP` was issued, or when it was completed. `SNAPSHOTSTART` causes the backup to match the state of the cluster when the backup began; `SNAPSHOTEND` causes the backup to reflect the state of the cluster when the backup was finished. `SNAPSHOTEND` is the default, and matches the behavior found in previous NDB Cluster releases.

**Note**

If you use the `SNAPSHOTSTART` option with `START BACKUP`, and the `CompressedBackup` parameter is enabled, only the data and control files are compressed—the log file is not compressed.

If both a *`wait_option`* and a *`snapshot_option`* are used, they may be specified in either order. For example, all of the following commands are valid, assuming that there is no existing backup having 4 as its ID:

```
START BACKUP WAIT STARTED SNAPSHOTSTART
START BACKUP SNAPSHOTSTART WAIT STARTED
START BACKUP 4 WAIT COMPLETED SNAPSHOTSTART
START BACKUP SNAPSHOTEND WAIT COMPLETED
START BACKUP 4 NOWAIT SNAPSHOTSTART
```

The procedure for creating a backup consists of the following steps:

1. Start the management client (`ndb_mgm`), if it not running already.

2. Execute the **START BACKUP** command. This produces several lines of output indicating the progress of the backup, as shown here:

```
ndb_mgm> START BACKUP
Waiting for completed, this may take several minutes
Node 2: Backup 1 started from node 1
Node 2: Backup 1 started from node 1 completed
 StartGCP: 177 StopGCP: 180
 #Records: 7362 #LogRecords: 0
 Data: 453648 bytes Log: 0 bytes
ndb_mgm>
```

3. When the backup has started the management client displays this message:

```
Backup backup_id started from node node_id
```

*backup_id* is the unique identifier for this particular backup. This identifier is saved in the cluster log, if it has not been configured otherwise. *node_id* is the identifier of the management server that is coordinating the backup with the data nodes. At this point in the backup process the cluster has received and processed the backup request. It does not mean that the backup has finished. An example of this statement is shown here:

```
Node 2: Backup 1 started from node 1
```

4. The management client indicates with a message like this one that the backup has started:

```
Backup backup_id started from node node_id completed
```

As is the case for the notification that the backup has started, *backup_id* is the unique identifier for this particular backup, and *node_id* is the node ID of the management server that is coordinating the backup with the data nodes. This output is accompanied by additional information including relevant global checkpoints, the number of records backed up, and the size of the data, as shown here:

```
Node 2: Backup 1 started from node 1 completed
 StartGCP: 177 StopGCP: 180
 #Records: 7362 #LogRecords: 0
 Data: 453648 bytes Log: 0 bytes
```

It is also possible to perform a backup from the system shell by invoking `ndb_mgm` with the `-e` or `--execute` option, as shown in this example:

```
$> ndb_mgm -e "START BACKUP 6 WAIT COMPLETED SNAPSHOTSTART"
```

When using `START BACKUP` in this way, you must specify the backup ID.

Cluster backups are created by default in the `BACKUP` subdirectory of the `DataDir` on each data node. This can be overridden for one or more data nodes individually, or for all cluster data nodes in the `config.ini` file using the `BackupDataDir` configuration parameter. The backup files created for a backup with a given *backup_id* are stored in a subdirectory named `BACKUP-`*backup_id* in the backup directory.

**Cancelling backups.** To cancel or abort a backup that is already in progress, perform the following steps:

1. Start the management client.

2. Execute this command:

```
ndb_mgm> ABORT BACKUP backup_id
```

The number *backup_id* is the identifier of the backup that was included in the response of the management client when the backup was started (in the message `Backup` *backup_id* `started from node` *management_node_id*).

3. The management client acknowledges the abort request with `Abort of backup backup_id ordered`.

**Note**

At this point, the management client has not yet received a response from the cluster data nodes to this request, and the backup has not yet actually been aborted.

4. After the backup has been aborted, the management client reports this fact in a manner similar to what is shown here:

```
Node 1: Backup 3 started from 5 has been aborted.
  Error: 1321 - Backup aborted by user request: Permanent error: User defined error
Node 3: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 2: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
Node 4: Backup 3 started from 5 has been aborted.
  Error: 1323 - 1323: Permanent error: Internal error
```

In this example, we have shown sample output for a cluster with 4 data nodes, where the sequence number of the backup to be aborted is 3, and the management node to which the cluster management client is connected has the node ID 5. The first node to complete its part in aborting the backup reports that the reason for the abort was due to a request by the user. (The remaining nodes report that the backup was aborted due to an unspecified internal error.)

**Note**

There is no guarantee that the cluster nodes respond to an `ABORT BACKUP` command in any particular order.

The `Backup backup_id started from node management_node_id has been aborted` messages mean that the backup has been terminated and that all files relating to this backup have been removed from the cluster file system.

It is also possible to abort a backup in progress from a system shell using this command:

```
$> ndb_mgm -e "ABORT BACKUP backup_id"
```

**Note**

If there is no backup having the ID *`backup_id`* running when an `ABORT BACKUP` is issued, the management client makes no response, nor is it indicated in the cluster log that an invalid abort command was sent.

### 23.6.8.3 Configuration for NDB Cluster Backups

Five configuration parameters are essential for backup:

- `BackupDataBufferSize`

  The amount of memory used to buffer data before it is written to disk.

- `BackupLogBufferSize`

  The amount of memory used to buffer log records before these are written to disk.

- `BackupMemory`

  The total memory allocated in a data node for backups. This should be the sum of the memory allocated for the backup data buffer and the backup log buffer.

- `BackupWriteSize`

The default size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

- `BackupMaxWriteSize`

  The maximum size of blocks written to disk. This applies for both the backup data buffer and the backup log buffer.

In addition, `CompressedBackup` causes `NDB` to use compression when creating and writing to backup files.

More detailed information about these parameters can be found in Backup Parameters.

You can also set a location for the backup files using the `BackupDataDir` configuration parameter. The default is `FileSystemPath/BACKUP/BACKUP-backup_id`.

In NDB 8.0.22 and later, you can enforce encryption of backup files by enabling `RequireEncryptedBackup`. When this parameter is set to 1, backups cannot be created without specifying `ENCRYPT PASSWORD=password` as part of a `START BACKUP` command.

### 23.6.8.4 NDB Cluster Backup Troubleshooting

If an error code is returned when issuing a backup request, the most likely cause is insufficient memory or disk space. You should check that there is enough memory allocated for the backup.

**Important**

If you have set `BackupDataBufferSize` and `BackupLogBufferSize` and their sum is greater than 4MB, then you must also set `BackupMemory` as well.

You should also make sure that there is sufficient space on the hard drive partition of the backup target.

`NDB` does not support repeatable reads, which can cause problems with the restoration process. Although the backup process is “hot”, restoring an NDB Cluster from backup is not a 100% “hot” process. This is due to the fact that, for the duration of the restore process, running transactions get nonrepeatable reads from the restored data. This means that the state of the data is inconsistent while the restore is in progress.

### 23.6.8.5 Taking an NDB Backup with Parallel Data Nodes

It is possible in NDB 8.0 to take a backup with multiple local data managers (LDMs) acting in parallel on the data nodes. For this to work, all data nodes in the cluster must use multiple LDMs, and each data node must use the same number of LDMs. This means that all data nodes must run `ndbmtd` (`ndbd` is single-threaded and thus always has only one LDM) and they must be configured to use multiple LDMs before taking the backup; `ndbmtd` by default runs in single-threaded mode. You can cause them to use multiple LDMs by choosing an appropriate setting for one of the multi-threaded data node configuration parameters `MaxNoOfExecutionThreads` or `ThreadConfig`. Keep in mind that changing these parameters requires a restart of the cluster; this can be a rolling restart. In addition, the `EnableMultithreadedBackup` parameter must be set to 1 for each data node (this is the default).

Depending on the number of LDMs and other factors, you may also need to increase `NoOfFragmentLogParts`. If you are using large Disk Data tables, you may also need to increase `DiskPageBufferMemory`. As with single-threaded backups, you may also want or need to make adjustments to settings for `BackupDataBufferSize`, `BackupMemory`, and other configuration parameters relating to backups (see Backup parameters).

Once all data nodes are using multiple LDMs, you can take the parallel backup using the `START BACKUP` command in the NDB management client just as you would if the data nodes were running

ndbd (or ndbmtd in single-threaded mode); no additional or special syntax is required, and you can specify a backup ID, wait option, or snapshot option in any combination as needed or desired.

Backups using multiple LDMs create subdirectories, one per LDM, under the directory BACKUP/BACKUP-*backup_id*/ (which in turn resides under the BackupDataDir) on each data node; these subdirectories are named BACKUP-*backup_id*-PART-1-OF-*N*/, BACKUP-*backup_id*-PART-2-OF-*N*/, and so on, up to BACKUP-*backup_id*-PART-*N*-OF-*N*/, where *backup_id* is the backup ID used for this backup and *N* is the number of LDMs per data node. Each of these subdirectories contains the usual backup files BACKUP-*backup_id*-0.*node_id*.Data, BACKUP-*backup_id*.*node_id*.ctl, and BACKUP-backup_id.node_id.log, where *node_id* is the node ID of this data node.

ndb_restore automatically checks for the presence of the subdirectories just described; if it finds them, it attempts to restore the backup in parallel. For information about restoring backups taken with multiple LDMs, see Section 23.5.23.3, “Restoring from a backup taken in parallel”.

To force creation of a single-threaded backup that can easily be imported by ndb_restore from an NDB release prior to 8.0, you can set EnableMultithreadedBackup = 0 for all data nodes (you can do this by setting the parameter in the [ndbd default] section of the config.ini global configuration file). It is also possible to restore a parallel backup to a cluster running an older version of NDB. See Restoring an NDB backup to a previous version of NDB Cluster, for more information.

## 23.6.9 Importing Data Into MySQL Cluster

It is common when setting up a new instance of NDB Cluster to need to import data from an existing NDB Cluster, instance of MySQL, or other source. This data is most often available in one or more of the following formats:

- An SQL dump file such as produced by mysqldump or mysqlpump. This can be imported using the mysql client, as shown later in this section.
- A CSV file produced by mysqldump or other export program. Such files can be imported into NDB using LOAD DATA INFILE in the mysql client, or with the ndb_import utility provided with the NDB Cluster distribution. For more information about the latter, see Section 23.5.13, “ndb_import — Import CSV Data Into NDB”.
- A native NDB backup produced using START BACKUP in the NDB management client. To import a native backup, you must use the ndb_restore program that comes as part of NDB Cluster. See Section 23.5.23, “ndb_restore — Restore an NDB Cluster Backup”, for more about using this program.

When importing data from an SQL file, it is often not necessary to enforce transactions or foreign keys, and temporarily disabling these features can speed up the import process greatly. This can be done using the mysql client, either from a client session, or by invoking it on the command line. Within a mysql client session, you can perform the import using the following SQL statements:

```
SET ndb_use_transactions=0;
SET foreign_key_checks=0;

source path/to/dumpfile;

SET ndb_use_transactions=1;
SET foreign_key_checks=1;
```

When performing the import in this fashion, you *must* enable ndb_use_transaction and foreign_key_checks again following execution of the mysql client's source command. Otherwise, it is possible for later statements in same session may also be executed without enforcing transactions or foreign key constraints, and which could lead to data inconcsistency.

From the system shell, you can import the SQL file while disabling enforcement of transaction and foreign keys by using the mysql client with the --init-command option, like this:

```
$> mysql --init-command='SET ndb_use_transactions=0; SET foreign_key_checks=0' < path/to/dumpfile
```

It is also possible to load the data into an `InnoDB` table, and convert it to use the NDB storage engine afterwards using ALTER TABLE ... ENGINE NDB). You should take into account, especially for many tables, that this may require a number of such operations; in addition, if foreign keys are used, you must mind the order of the `ALTER TABLE` statements carefully, due to the fact that foreign keys do not work between tables using different MySQL storage engines.

You should be aware that the methods described previously in this section are not optimized for very large data sets or large transactions. Should an application really need big transactions or many concurrent transactions as part of normal operation, you may wish to increase the value of the `MaxNoOfConcurrentOperations` data node configuration parameter, which reserves more memory to allow a data node to take over a transaction if its transaction coordinator stops unexpectedly.

You may also wish to do this when performing bulk `DELETE` or `UPDATE` operations on NDB Cluster tables. If possible, try to have applications perform these operations in chunks, for example, by adding `LIMIT` to such statements.

If a data import operation does not complete successfully, for whatever reason, you should be prepared to perform any necessary cleanup including possibly one or more `DROP TABLE` statements, `DROP DATABASE` statements, or both. Failing to do so may leave the database in an inconsistent state.

## 23.6.10 MySQL Server Usage for NDB Cluster

`mysqld` is the traditional MySQL server process. To be used with NDB Cluster, `mysqld` needs to be built with support for the `NDB` storage engine, as it is in the precompiled binaries available from https://dev.mysql.com/downloads/. If you build MySQL from source, you must invoke `CMake` with the `-DWITH_NDB=1` or (deprecated) `-DWITH_NDBCLUSTER=1` option to include support for `NDB`.

For more information about compiling NDB Cluster from source, see Section 23.3.1.4, “Building NDB Cluster from Source on Linux”, and Section 23.3.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

(For information about `mysqld` options and variables, in addition to those discussed in this section, which are relevant to NDB Cluster, see Section 23.4.3.9, “MySQL Server Options and Variables for NDB Cluster”.)

If the `mysqld` binary has been built with Cluster support, the `NDBCLUSTER` storage engine is still disabled by default. You can use either of two possible options to enable this engine:

- Use `--ndbcluster` as a startup option on the command line when starting `mysqld`.
- Insert a line containing `ndbcluster` in the `[mysqld]` section of your `my.cnf` file.

An easy way to verify that your server is running with the `NDBCLUSTER` storage engine enabled is to issue the `SHOW ENGINES` statement in the MySQL Monitor (`mysql`). You should see the value `YES` as the `Support` value in the row for `NDBCLUSTER`. If you see `NO` in this row or if there is no such row displayed in the output, you are not running an `NDB`-enabled version of MySQL. If you see `DISABLED` in this row, you need to enable it in either one of the two ways just described.

To read cluster configuration data, the MySQL server requires at a minimum three pieces of information:

- The MySQL server's own cluster node ID
- The host name or IP address for the management server
- The number of the TCP/IP port on which it can connect to the management server

Node IDs can be allocated dynamically, so it is not strictly necessary to specify them explicitly.

The `mysqld` parameter `ndb-connectstring` is used to specify the connection string either on the command line when starting `mysqld` or in `my.cnf`. The connection string contains the host name or IP address where the management server can be found, as well as the TCP/IP port it uses.

In the following example, `ndb_mgmd.mysql.com` is the host where the management server resides, and the management server listens for cluster messages on port 1186:

```
$> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 23.4.3.3, “NDB Cluster Connection Strings”, for more information on connection strings.

Given this information, the MySQL server can act as a full participant in the cluster. (We often refer to a `mysqld` process running in this manner as an SQL node.) It is fully aware of all cluster data nodes as well as their status, and establishes connections to all data nodes. In this case, it is able to use any data node as a transaction coordinator and to read and update node data.

You can see in the `mysql` client whether a MySQL server is connected to the cluster using `SHOW PROCESSLIST`. If the MySQL server is connected to the cluster, and you have the `PROCESS` privilege, then the first row of the output is as shown here:

```
mysql> SHOW PROCESSLIST \G
*************************** 1. row ***************************
     Id: 1
   User: system user
   Host:
     db:
Command: Daemon
   Time: 1
  State: Waiting for event from ndbcluster
   Info: NULL
```

**Important**

To participate in an NDB Cluster, the `mysqld` process must be started with *both* the options `--ndbcluster` and `--ndb-connectstring` (or their equivalents in `my.cnf`). If `mysqld` is started with only the `--ndbcluster` option, or if it is unable to contact the cluster, it is not possible to work with `NDB` tables, *nor is it possible to create any new tables regardless of storage engine*. The latter restriction is a safety measure intended to prevent the creation of tables having the same names as `NDB` tables while the SQL node is not connected to the cluster. If you wish to create tables using a different storage engine while the `mysqld` process is not participating in an NDB Cluster, you must restart the server *without* the `--ndbcluster` option.

## 23.6.11 NDB Cluster Disk Data Tables

NDB Cluster supports storing nonindexed columns of `NDB` tables on disk, rather than in RAM. Column data and logging metadata are kept in data files and undo log files, conceptualized as tablespaces and log file groups, as described in the next section—see Section 23.6.11.1, “NDB Cluster Disk Data Objects”.

NDB Cluster Disk Data performance can be influenced by a number of configuration parameters. For information about these parameters and their effects, see Disk Data Configuration Parameters, and Disk Data and GCP Stop errors.

You should also set the `DiskDataUsingSameDisk` data node configuration parameter to `false` when using separate disks for Disk Data files.

See also Disk Data file system parameters.

NDB 8.0 provides improved support when using Disk Data tables with solid-state drives, in particular those using NVMe. See the following documentation for more information:

- Disk Data latency parameters
- Section 23.6.16.31, “The ndbinfo diskstat Table”

- Section 23.6.16.32, “The ndbinfo diskstats_1sec Table”
- Section 23.6.16.49, “The ndbinfo pgman_time_track_stats Table”

### 23.6.11.1 NDB Cluster Disk Data Objects

NDB Cluster Disk Data storage is implemented using the following objects:

- *Tablespace*: Acts as containers for other Disk Data objects. A tablespace contains one or more data files and one or more undo log file groups.
- *Data file*: Stores column data. A data file is assigned directly to a tablespace.
- *Undo log file*: Contains undo information required for rolling back transactions. Assigned to an undo log file group.
- *log file group*: Contains one or more undo log files. Assigned to a tablespace.

Undo log files and data files are actual files in the file system of each data node; by default they are placed in `ndb_node_id_fs` in the `DataDir` specified in the NDB Cluster `config.ini` file, and where `node_id` is the data node's node ID. It is possible to place these elsewhere by specifying either an absolute or relative path as part of the filename when creating the undo log or data file. Statements that create these files are shown later in this section.

Undo log files are used only by Disk Data tables, and are not needed or used by `NDB` tables that are stored in memory only.

NDB Cluster tablespaces and log file groups are not implemented as files.

Although not all Disk Data objects are implemented as files, they all share the same namespace. This means that *each Disk Data object* must be uniquely named (and not merely each Disk Data object of a given type). For example, you cannot have a tablespace and a log file group both named `dd1`.

Assuming that you have already set up an NDB Cluster with all nodes (including management and SQL nodes), the basic steps for creating an NDB Cluster table on disk are as follows:

1. Create a log file group, and assign one or more undo log files to it (an undo log file is also sometimes referred to as an *undofile*).
2. Create a tablespace; assign the log file group, as well as one or more data files, to the tablespace.
3. Create a Disk Data table that uses this tablespace for data storage.

Each of these tasks can be accomplished using SQL statements in the `mysql` client or other MySQL client application, as shown in the example that follows.

1. We create a log file group named `lg_1` using `CREATE LOGFILE GROUP`. This log file group is to be made up of two undo log files, which we name `undo_1.log` and `undo_2.log`, whose initial sizes are 16 MB and 12 MB, respectively. (The default initial size for an undo log file is 128 MB.) Optionally, you can also specify a size for the log file group's undo buffer, or permit it to assume the default value of 8 MB. In this example, we set the UNDO buffer's size at 2 MB. A log file group must be created with an undo log file; so we add `undo_1.log` to `lg_1` in this `CREATE LOGFILE GROUP` statement:

```
CREATE LOGFILE GROUP lg_1
    ADD UNDOFILE 'undo_1.log'
    INITIAL_SIZE 16M
    UNDO_BUFFER_SIZE 2M
    ENGINE NDBCLUSTER;
```

To add `undo_2.log` to the log file group, use the following `ALTER LOGFILE GROUP` statement:

```
ALTER LOGFILE GROUP lg_1
```

```
    ADD UNDOFILE 'undo_2.log'
    INITIAL_SIZE 12M
    ENGINE NDBCLUSTER;
```

Some items of note:

- The `.log` file extension used here is not required. We employ it merely to make the log files easily recognizable.
- Every `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statement must include an `ENGINE` option. The only permitted values for this option are `NDBCLUSTER` and `NDB`.

**Important**

There can exist at most one log file group in the same NDB Cluster at any given time.

- When you add an undo log file to a log file group using `ADD UNDOFILE 'filename'`, a file with the name *`filename`* is created in the `ndb_node_id_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each undo log file is of the size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER LOGFILE GROUP` statement just shown creates 4 undo log files, 1 each on in the data directory of each of the 4 data nodes; each of these files is named `undo_2.log` and each file is 12 MB in size.
- `UNDO_BUFFER_SIZE` is limited by the amount of system memory available.
- See Section 13.1.16, “CREATE LOGFILE GROUP Statement”, and Section 13.1.6, “ALTER LOGFILE GROUP Statement”, for more information about these statements.

2. Now we can create a tablespace—an abstract container for files used by Disk Data tables to store data. A tablespace is associated with a particular log file group; when creating a new tablespace, you must specify the log file group it uses for undo logging. You must also specify at least one data file; you can add more data files to the tablespace after the tablespace is created. It is also possible to drop data files from a tablespace (see example later in this section).

   Assume that we wish to create a tablespace named `ts_1` which uses `lg_1` as its log file group. We want the tablespace to contain two data files, named `data_1.dat` and `data_2.dat`, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for `INITIAL_SIZE` is 128 MB.) We can do this using two SQL statements, as shown here:

```
CREATE TABLESPACE ts_1
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP lg_1
    INITIAL_SIZE 32M
    ENGINE NDBCLUSTER;

ALTER TABLESPACE ts_1
    ADD DATAFILE 'data_2.dat'
    INITIAL_SIZE 48M;
```

   The `CREATE TABLESPACE` statement creates a tablespace `ts_1` with the data file `data_1.dat`, and associates `ts_1` with log file group `lg_1`. The `ALTER TABLESPACE` adds the second data file (`data_2.dat`).

   Some items of note:

   - As is the case with the `.log` file extension used in this example for undo log files, there is no special significance for the `.dat` file extension; it is used merely for easy recognition.
   - When you add a data file to a tablespace using `ADD DATAFILE 'filename'`, a file with the name *`filename`* is created in the `ndb_node_id_fs` directory within the `DataDir` of each data node in the cluster, where *`node_id`* is the node ID of the data node. Each data file is of the

size specified in the SQL statement. For example, if an NDB Cluster has 4 data nodes, then the `ALTER TABLESPACE` statement just shown creates 4 data files, 1 each in the data directory of each of the 4 data nodes; each of these files is named `data_2.dat` and each file is 48 MB in size.

- `NDB` reserves 4% of each tablespace for use during data node restarts. This space is not available for storing data.
- `CREATE TABLESPACE` statements must contain an `ENGINE` clause; only tables using the same storage engine as the tablespace can be created in the tablespace. For `ALTER TABLESPACE`, an `ENGINE` clause is accepted but is deprecated and subject to removal in a future release. For `NDB` tablespaces, the only permitted values for this option are `NDBCLUSTER` and `NDB`.
- In NDB 8.0.20 and later, allocation of extents is performed in round-robin fashion among all data files used by a given tablespace.
- For more information about the `CREATE TABLESPACE` and `ALTER TABLESPACE` statements, see Section 13.1.21, “CREATE TABLESPACE Statement”, and Section 13.1.10, “ALTER TABLESPACE Statement”.

3. Now it is possible to create a table whose unindexed columns are stored on disk using files in tablespace `ts_1`:

```
CREATE TABLE dt_1 (
    member_id INT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    last_name VARCHAR(50) NOT NULL,
    first_name VARCHAR(50) NOT NULL,
    dob DATE NOT NULL,
    joined DATE NOT NULL,
    INDEX(last_name, first_name)
    )
    TABLESPACE ts_1 STORAGE DISK
    ENGINE NDBCLUSTER;
```

`TABLESPACE ts_1 STORAGE DISK` tells the `NDB` storage engine to use tablespace `ts_1` for data storage on disk.

Once table `ts_1` has been created as shown, you can perform `INSERT`, `SELECT`, `UPDATE`, and `DELETE` statements on it just as you would with any other MySQL table.

It is also possible to specify whether an individual column is stored on disk or in memory by using a `STORAGE` clause as part of the column's definition in a `CREATE TABLE` or `ALTER TABLE` statement. `STORAGE DISK` causes the column to be stored on disk, and `STORAGE MEMORY` causes in-memory storage to be used. See Section 13.1.20, “CREATE TABLE Statement”, for more information.

You can obtain information about the `NDB` disk data files and undo log files just created by querying the `FILES` table in the `INFORMATION_SCHEMA` database, as shown here:

```
mysql> SELECT
             FILE_NAME AS File, FILE_TYPE AS Type,
             TABLESPACE_NAME AS Tablespace, TABLE_NAME AS Name,
             LOGFILE_GROUP_NAME AS 'File group',
             FREE_EXTENTS AS Free, TOTAL_EXTENTS AS Total
          FROM INFORMATION_SCHEMA.FILES
          WHERE ENGINE='ndbcluster';
+-------------+----------+------------+------+------------+------+---------+
| File        | Type     | Tablespace | Name | File group | Free | Total   |
+-------------+----------+------------+------+------------+------+---------+
| ./undo_1.log | UNDO LOG | lg_1       | NULL | lg_1       |    0 | 4194304 |
| ./undo_2.log | UNDO LOG | lg_1       | NULL | lg_1       |    0 | 3145728 |
| ./data_1.dat | DATAFILE | ts_1       | NULL | lg_1       |   32 |      32 |
| ./data_2.dat | DATAFILE | ts_1       | NULL | lg_1       |   48 |      48 |
+-------------+----------+------------+------+------------+------+---------+
4 rows in set (0.00 sec)
```

For more information and examples, see Section 26.3.15, “The INFORMATION_SCHEMA FILES Table”.

**Indexing of columns implicitly stored on disk.** For table `dt_1` as defined in the example just shown, only the `dob` and `joined` columns are stored on disk. This is because there are indexes on the `id`, `last_name`, and `first_name` columns, and so data belonging to these columns is stored in RAM. Only nonindexed columns can be held on disk; indexes and indexed column data continue to be stored in memory. This tradeoff between the use of indexes and conservation of RAM is something you must keep in mind as you design Disk Data tables.

You cannot add an index to a column that has been explicitly declared `STORAGE DISK`, without first changing its storage type to `MEMORY`; any attempt to do so fails with an error. A column which *implicitly* uses disk storage can be indexed; when this is done, the column's storage type is changed to `MEMORY` automatically. By “implicitly”, we mean a column whose storage type is not declared, but which is which inherited from the parent table. In the following CREATE TABLE statement (using the tablespace `ts_1` defined previously), columns `c2` and `c3` use disk storage implicitly:

```
mysql> CREATE TABLE ti (
    ->     c1 INT PRIMARY KEY,
    ->     c2 INT,
    ->     c3 INT,
    ->     c4 INT
    -> )
    ->     STORAGE DISK
    ->     TABLESPACE ts_1
    ->     ENGINE NDBCLUSTER;
Query OK, 0 rows affected (1.31 sec)
```

Because `c2`, `c3`, and `c4` are themselves not declared with `STORAGE DISK`, it is possible to index them. Here, we add indexes to `c2` and `c3`, using, respectively, `CREATE INDEX` and `ALTER TABLE`:

```
mysql> CREATE INDEX i1 ON ti(c2);
Query OK, 0 rows affected (2.72 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> ALTER TABLE ti ADD INDEX i2(c3);
Query OK, 0 rows affected (0.92 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

`SHOW CREATE TABLE` confirms that the indexes were added.

```
mysql> SHOW CREATE TABLE ti\G
*************************** 1. row ***************************
       Table: ti
Create Table: CREATE TABLE `ti` (
  `c1` int(11) NOT NULL,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  `c4` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`),
  KEY `i1` (`c2`),
  KEY `i2` (`c3`)
) /*!50100 TABLESPACE `ts_1` STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

You can see using `ndb_desc` that the indexed columns (emphasized text) now use in-memory rather than on-disk storage:

```
$> ./ndb_desc -d test t1
-- t1 --
Version: 33554433
Fragment type: HashMapPartition
K Value: 6
Min load factor: 78
Max load factor: 80
Temporary table: no
Number of attributes: 4
```

```
Number of primary keys: 1
Length of frm data: 317
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 4
FragmentCount: 4
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-4
-- Attributes --
c1 Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY
c2 Int NULL AT=FIXED ST=MEMORY
c3 Int NULL AT=FIXED ST=MEMORY
c4 Int NULL AT=FIXED ST=DISK
-- Indexes --
PRIMARY KEY(c1) - UniqueHashIndex
i2(c3) - OrderedIndex
PRIMARY(c1) - OrderedIndex
i1(c2) - OrderedIndex

NDBT_ProgramExit: 0 - OK
```

**Performance note.** The performance of a cluster using Disk Data storage is greatly improved if Disk Data files are kept on a separate physical disk from the data node file system. This must be done for each data node in the cluster to derive any noticeable benefit.

You can use absolute and relative file system paths with `ADD UNDOFILE` and `ADD DATAFILE`; relative paths are calculated with respect to the data node's data directory.

A log file group, a tablespace, and any Disk Data tables using these must be created in a particular order. This is also true for dropping these objects, subject to the following constraints:

- A log file group cannot be dropped as long as any tablespaces use it.
- A tablespace cannot be dropped as long as it contains any data files.
- You cannot drop any data files from a tablespace as long as there remain any tables which are using the tablespace.
- It is not possible to drop files created in association with a different tablespace other than the one with which the files were created.

For example, to drop all the objects created so far in this section, you can use the following statements:

```
mysql> DROP TABLE dt_1;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_2.dat'
    -> ENGINE NDBCLUSTER;

mysql> ALTER TABLESPACE ts_1
    -> DROP DATAFILE 'data_1.dat'
    -> ENGINE NDBCLUSTER;

mysql> DROP TABLESPACE ts_1
    -> ENGINE NDBCLUSTER;

mysql> DROP LOGFILE GROUP lg_1
    -> ENGINE NDBCLUSTER;
```

These statements must be performed in the order shown, except that the two `ALTER TABLESPACE ... DROP DATAFILE` statements may be executed in either order.

### 23.6.11.2 NDB Cluster Disk Data Storage Requirements

The following items apply to Disk Data storage requirements:

- Variable-length columns of Disk Data tables take up a fixed amount of space. For each row, this is equal to the space required to store the largest possible value for that column.

  For general information about calculating these values, see Section 11.7, “Data Type Storage Requirements”.

  You can obtain an estimate the amount of space available in data files and undo log files by querying the Information Schema `FILES` table. For more information and examples, see Section 26.3.15, “The INFORMATION_SCHEMA FILES Table”.

  **Note**

  The `OPTIMIZE TABLE` statement does not have any effect on Disk Data tables.

- In a Disk Data table, the first 256 bytes of a `TEXT` or `BLOB` column are stored in memory; only the remainder is stored on disk.

- Each row in a Disk Data table uses 8 bytes in memory to point to the data stored on disk. This means that, in some cases, converting an in-memory column to the disk-based format can actually result in greater memory usage. For example, converting a `CHAR(4)` column from memory-based to disk-based format increases the amount of `DataMemory` used per row from 4 to 8 bytes.

  **Important**

  Starting the cluster with the `--initial` option does *not* remove Disk Data files. You must remove these manually prior to performing an initial restart of the cluster.

Performance of Disk Data tables can be improved by minimizing the number of disk seeks by making sure that `DiskPageBufferMemory` is of sufficient size. You can query the `diskpagebuffer` table to help determine whether the value for this parameter needs to be increased.

## 23.6.12 Online Operations with ALTER TABLE in NDB Cluster

MySQL NDB Cluster 8.0 supports online table schema changes using `ALTER TABLE ... ALGORITHM=DEFAULT|INPLACE|COPY`. NDB Cluster handles `COPY` and `INPLACE` as described in the next few paragraphs.

For `ALGORITHM=COPY`, the `mysqld` NDB Cluster handler performs the following actions:

- Tells the data nodes to create an empty copy of the table, and to make the required schema changes to this copy.
- Reads rows from the original table, and writes them to the copy.
- Tells the data nodes to drop the original table and then to rename the copy.

We sometimes refer to this as a “copying” or “offline” `ALTER TABLE`.

DML operations are not permitted concurrently with a copying `ALTER TABLE`.

The `mysqld` on which the copying `ALTER TABLE` statement is issued takes a metadata lock, but this is in effect only on that `mysqld`. Other `NDB` clients can modify row data during a copying `ALTER TABLE`, resulting in inconsistency.

For `ALGORITHM=INPLACE`, the NDB Cluster handler tells the data nodes to make the required changes, and does not perform any copying of data.

We also refer to this as a “non-copying” or “online” `ALTER TABLE`.

A non-copying `ALTER TABLE` allows concurrent DML operations.

`ALGORITHM=INSTANT` is not supported by NDB 8.0.

Regardless of the algorithm used, the `mysqld` takes a Global Schema Lock (GSL) while executing `ALTER TABLE`; this prevents execution of any (other) DDL or backups concurrently on this or any other SQL node in the cluster. This is normally not problematic, unless the `ALTER TABLE` takes a very long time.

**Note**

Some older releases of NDB Cluster used a syntax specific to `NDB` for online `ALTER TABLE` operations. That syntax has since been removed.

Operations that add and drop indexes on variable-width columns of `NDB` tables occur online. Online operations are noncopying; that is, they do not require that indexes be re-created. They do not lock the table being altered from access by other API nodes in an NDB Cluster (but see Limitations of NDB online operations, later in this section). Such operations do not require single user mode for `NDB` table alterations made in an NDB cluster with multiple API nodes; transactions can continue uninterrupted during online DDL operations.

`ALGORITHM=INPLACE` can be used to perform online `ADD COLUMN`, `ADD INDEX` (including `CREATE INDEX` statements), and `DROP INDEX` operations on `NDB` tables. Online renaming of `NDB` tables is also supported (prior to NDB 8.0, such columns could not be renamed online).

Disk-based columns cannot be added to `NDB` tables online. This means that, if you wish to add an in-memory column to an `NDB` table that uses a table-level `STORAGE DISK` option, you must declare the new column as using memory-based storage explicitly. For example—assuming that you have already created tablespace `ts1`—suppose that you create table `t1` as follows:

```
mysql> CREATE TABLE t1 (
     >     c1 INT NOT NULL PRIMARY KEY,
     >     c2 VARCHAR(30)
     >     )
     >     TABLESPACE ts1 STORAGE DISK
     >     ENGINE NDB;
Query OK, 0 rows affected (1.73 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

You can add a new in-memory column to this table online as shown here:

```
mysql> ALTER TABLE t1
     >     ADD COLUMN c3 INT COLUMN_FORMAT DYNAMIC STORAGE MEMORY,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected (1.25 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

This statement fails if the `STORAGE MEMORY` option is omitted:

```
mysql> ALTER TABLE t1
     >     ADD COLUMN c4 INT COLUMN_FORMAT DYNAMIC,
     >     ALGORITHM=INPLACE;
ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported. Reason:
Adding column(s) or add/reorganize partition not supported online. Try
ALGORITHM=COPY.
```

If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically, but a warning is issued, as shown here:

```
mysql> ALTER ONLINE TABLE t1 ADD COLUMN c4 INT STORAGE MEMORY;
Query OK, 0 rows affected, 1 warning (1.17 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS\G
*************************** 1. row ***************************
  Level: Warning
```

```
   Code: 1478
Message: DYNAMIC column c4 with STORAGE DISK is not supported, column will
become FIXED


mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t1` (
  `c1` int(11) NOT NULL,
  `c2` varchar(30) DEFAULT NULL,
  `c3` int(11) /*!50606 STORAGE MEMORY */ /*!50606 COLUMN_FORMAT DYNAMIC */ DEFAULT NULL,
  `c4` int(11) /*!50606 STORAGE MEMORY */ DEFAULT NULL,
  PRIMARY KEY (`c1`)
) /*!50606 TABLESPACE ts_1 STORAGE DISK */ ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.03 sec)
```

**Note**

The `STORAGE` and `COLUMN_FORMAT` keywords are supported only in NDB Cluster; in any other version of MySQL, attempting to use either of these keywords in a `CREATE TABLE` or `ALTER TABLE` statement results in an error.

It is also possible to use the statement `ALTER TABLE ... REORGANIZE PARTITION, ALGORITHM=INPLACE` with no *`partition_names`* `INTO (`*`partition_definitions`*`)` option on `NDB` tables. This can be used to redistribute NDB Cluster data among new data nodes that have been added to the cluster online. This does *not* perform any defragmentation, which requires an `OPTIMIZE TABLE` or null `ALTER TABLE` statement. For more information, see Section 23.6.7, “Adding NDB Cluster Data Nodes Online”.

## Limitations of NDB online operations

Online `DROP COLUMN` operations are not supported.

Online `ALTER TABLE`, `CREATE INDEX`, or `DROP INDEX` statements that add columns or add or drop indexes are subject to the following limitations:

- A given online `ALTER TABLE` can use only one of `ADD COLUMN`, `ADD INDEX`, or `DROP INDEX`. One or more columns can be added online in a single statement; only one index may be created or dropped online in a single statement.
- The table being altered is not locked with respect to API nodes other than the one on which an online `ALTER TABLE ADD COLUMN`, `ADD INDEX`, or `DROP INDEX` operation (or `CREATE INDEX` or `DROP INDEX` statement) is run. However, the table is locked against any other operations originating on the *same* API node while the online operation is being executed.
- The table to be altered must have an explicit primary key; the hidden primary key created by the `NDB` storage engine is not sufficient for this purpose.
- The storage engine used by the table cannot be changed online.
- The tablespace used by the table cannot be changed online. Beginning with NDB 8.0.21, a statement such as `ALTER TABLE` *`ndb_table`* `... ALGORITHM=INPLACE, TABLESPACE=`*`new_tablespace`* is specifically disallowed. (Bug #99269, Bug #31180526)
- When used with NDB Cluster Disk Data tables, it is not possible to change the storage type (`DISK` or `MEMORY`) of a column online. This means, that when you add or drop an index in such a way that the operation would be performed online, and you want the storage type of the column or columns to be changed, you must use `ALGORITHM=COPY` in the statement that adds or drops the index.

Columns to be added online cannot use the `BLOB` or `TEXT` type, and must meet the following criteria:

- The columns must be dynamic; that is, it must be possible to create them using `COLUMN_FORMAT DYNAMIC`. If you omit the `COLUMN_FORMAT DYNAMIC` option, the dynamic column format is employed automatically.

- The columns must permit `NULL` values and not have any explicit default value other than `NULL`. Columns added online are automatically created as `DEFAULT NULL`, as can be seen here:

```
mysql> CREATE TABLE t2 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
     >     ADD COLUMN c2 INT,
     >     ADD COLUMN c3 INT,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW CREATE TABLE t1\G
*************************** 1. row ***************************
       Table: t1
Create Table: CREATE TABLE `t2` (
  `c1` int(11) NOT NULL AUTO_INCREMENT,
  `c2` int(11) DEFAULT NULL,
  `c3` int(11) DEFAULT NULL,
  PRIMARY KEY (`c1`)
) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)
```

- The columns must be added following any existing columns. If you attempt to add a column online before any existing columns or using the `FIRST` keyword, the statement fails with an error.
- Existing table columns cannot be reordered online.

For online `ALTER TABLE` operations on `NDB` tables, fixed-format columns are converted to dynamic when they are added online, or when indexes are created or dropped online, as shown here (repeating the `CREATE TABLE` and `ALTER TABLE` statements just shown for the sake of clarity):

```
mysql> CREATE TABLE t2 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)

mysql> ALTER TABLE t2
     >     ADD COLUMN c2 INT,
     >     ADD COLUMN c3 INT,
     >     ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
*************************** 2. row ***************************
  Level: Warning
   Code: 1478
Message: Converted FIXED field 'c3' to DYNAMIC to enable online ADD COLUMN
2 rows in set (0.00 sec)
```

Only the column or columns to be added online must be dynamic. Existing columns need not be; this includes the table's primary key, which may also be `FIXED`, as shown here:

```
mysql> CREATE TABLE t3 (
     >     c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY COLUMN_FORMAT FIXED
     >     ) ENGINE=NDB;
Query OK, 0 rows affected (2.10 sec)

mysql> ALTER TABLE t3 ADD COLUMN c2 INT, ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql> SHOW WARNINGS;
*************************** 1. row ***************************
  Level: Warning
```

```
   Code: 1478
Message: Converted FIXED field 'c2' to DYNAMIC to enable online ADD COLUMN
1 row in set (0.00 sec)
```

Columns are not converted from `FIXED` to `DYNAMIC` column format by renaming operations. For more information about `COLUMN_FORMAT`, see Section 13.1.20, “CREATE TABLE Statement”.

The `KEY`, `CONSTRAINT`, and `IGNORE` keywords are supported in `ALTER TABLE` statements using `ALGORITHM=INPLACE`.

Setting `MAX_ROWS` to 0 using an online `ALTER TABLE` statement is disallowed. You must use a copying `ALTER TABLE` to perform this operation. (Bug #21960004)

## 23.6.13 Privilege Synchronization and NDB_STORED_USER

NDB 8.0 introduces a new mechanism for sharing and synchronizing users, roles, and privileges between SQL nodes connected to an NDB Cluster. This can be enabled by granting the `NDB_STORED_USER` privilege. See the description of the privilege for usage information.

`NDB_STORED_USER` is printed in the output of `SHOW GRANTS` as with any other privilege, as shown here:

```
mysql> SHOW GRANTS for 'jon'@'localhost';
+-----------------------------------------------------+
| Grants for jon@localhost                            |
+-----------------------------------------------------+
| GRANT USAGE ON *.* TO `jon`@`localhost`             |
| GRANT NDB_STORED_USER ON *.* TO `jon`@`localhost` |
+-----------------------------------------------------+
```

You can also verify that privileges are shared for this account using the `ndb_select_all` utility supplied with NDB Cluster, like this (some output wrapped to preserve formatting):

```
$> ndb_select_all -d mysql ndb_sql_metadata | grep '`jon`@`localhost`'
12      "'jon'@'localhost'"     0       [NULL]  "GRANT USAGE ON *.* TO `jon`@`localhost`"
11      "'jon'@'localhost'"     0       2       "CREATE USER `jon`@`localhost`
IDENTIFIED WITH 'caching_sha2_password' AS
0x24412430303524660143402 25A107D590E6E653B5D587922306102716D752E6656772F3038512F
6C5072776D30376D37347A384B557A4C564F70495158656A31382E45324E33
REQUIRE NONE PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK PASSWORD HISTORY DEFAULT
PASSWORD REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT"
12      "'jon'@'localhost'"     1       [NULL]  "GRANT NDB_STORED_USER ON *.* TO `jon`@`localhost`"
```

`ndb_sql_metadata` is a special `NDB` table that is not visible using the `mysql` or other MySQL client.

A statement granting the `NDB_STORED_USER` privilege, such as `GRANT NDB_STORED_USER ON *.* TO 'cluster_app_user'@'localhost'`, works by directing `NDB` to create a snapshot using the queries `SHOW CREATE USER cluster_app_user@localhost` and `SHOW GRANTS FOR cluster_app_user@localhost`, then storing the results in `ndb_sql_metadata`. Any other SQL nodes are then requested to read and apply the snapshot. Whenever a MySQL server starts up and joins the cluster as an SQL node it executes these stored `CREATE USER` and `GRANT` statements as part of the cluster schema synchronization process.

Whenever an SQL statement is executed on an SQL node other than the one where it originated, the statement is run in a utility thread of the `NDBCLUSTER` storage engine; this is done within a security environment equivalent to that of the MySQL replication replica applier thread.

Beginning with NDB 8.0.27, an SQL node performing a change to user privileges takes a global lock before doing so, which prevents deadlocks by concurrent ACL operations on different SQL nodes. Prior to NDB 8.0.27, changes to users with `NDB_STORED_USER` were updated in a completely asynchronous fashion, without any locks being taken.

You should keep in mind that, because shared schema change operations are performed synchronously, the next shared schema change following a change to any shared user or users serves as a synchronization point. Any pending user changes run to completion before the schema change distribution can begin; after this the schema change itself runs synchronously. For example, if a `DROP`

DATABASE statement follows a DROP USER of a distributed user, the drop of the database cannot take place until the drop of the user has completed on all SQL nodes.

In the event that multiple GRANT, REVOKE, or other user administration statements from multiple SQL nodes cause privileges for a given user to diverge on different SQL nodes, you can fix this problem by issuing GRANT NDB_STORED_USER for this user on an SQL node where the privileges are known to be correct; this causes a new snapshot of the privileges to be taken and synchronized to the other SQL nodes.

NDB Cluster 8.0 does not support distribution of MySQL users and privileges across SQL nodes in an NDB Cluster by altering the MySQL privilege tables such that they used the NDB storage engine as in NDB 7.6 and earlier releases (see Distributed Privileges Using Shared Grant Tables). For information about the impact of this change on upgrades to NDB 8.0 from a previous release, see Section 23.3.7, "Upgrading and Downgrading NDB Cluster".

## 23.6.14 File System Encryption for NDB Cluster

The following sections provide information about NDB data node file system encryption, as implemented in NDB 8.0.31 and later.

### 23.6.14.1 NDB File System Encryption Setup and Usage

*Encryption of file system*: To enable encryption of a previously unencrypted file system, the following steps are required:

1. Set the required data node parameters in the [ndbd default] section of the config.ini file, as shown here:

   ```
   [ndbd default]
   EncryptedFileSystem= 1
   ```

   These parameters must be set as shown on all data nodes.

2. Start the management server with either --initial or --reload to cause it to read the updated configuration file.

3. Perform a rolling initial start (or restart) of all the data nodes (see Section 23.6.5, "Performing a Rolling Restart of an NDB Cluster"): Start each data node with --initial; in addition, supply either of the options --filesystem-password or --filesystem-password-from-stdin, plus a password, to each data node process. When you supply the password on the command line, a warning is shown, similar to this one:

   ```
   > ndbmtd -c 127.0.0.1 --filesystem-password=ndbsecret
   ndbmtd: [Warning] Using a password on the command line interface can be insecure.
   2022-08-22 16:17:58 [ndbd] INFO     -- Angel connected to '127.0.0.1:1186'
   2022-08-22 16:17:58 [ndbd] INFO     -- Angel allocated nodeid: 5
   ```

   --filesystem-password can accept the password form a file, tty, or stdin; --filesystem-password-from-stdin accepts the password from stdin only. The latter protects the password from exposure on the process command line or in the file system, and allows for the possibility of passing it from another secure application.

   You can also place the password in a my.cnf file that can be read by the data node process, but not by other users of the system. Using the same password as in the previous example, the relevant portion of the file should look like this:

   ```
   [ndbd]

   filesystem-password=ndbsecret
   ```

   You can also prompt the user starting the data node process to supply the encryption password when doing so, by using the --filesystem-password-from-stdin option in the my.cnf file instead, like this:

```
[ndbd]

filesystem-password-from-stdin
```

In this case, the user is prompted for the password when starting the data node process, as shown here:

```
> ndbmtd -c 127.0.0.1
Enter filesystem password: *********
2022-08-22 16:36:00 [ndbd] INFO     -- Angel connected to '127.0.0.1:1186'
2022-08-22 16:36:00 [ndbd] INFO     -- Angel allocated nodeid: 5
>
```

Regardless of the method used, the format of the encryption password is the same as that used for passwords for encrypted backups (see Section 23.6.8.2, “Using The NDB Cluster Management Client to Create a Backup”); the password must be supplied when starting each data node process; otherwise the data node process cannot start. This is indicated by the following message in the data node log:

```
> tail -n2 ndb_5_out.log
2022-08-22 16:08:30 [ndbd] INFO     -- Data node configured to have encryption but password not provide
2022-08-22 16:08:31 [ndbd] ALERT    -- Node 5: Forced node shutdown completed. Occurred during startpha
```

When restarted as just described, each data node clears its on-disk state, and rebuilds it in encrypted form.

*Rotation of File system password*: To update the encryption password used by the data nodes, perform a rolling initial restart of the data nodes, supplying the new password to each data node when restarting it using `--filesystem-password` or `--filesystem-password-from-stdin`.

*Decryption of file system*: To remove encryption from an encrypted file system, do the following:

1. In the `[ndbd default]` section of the `config.ini` file, set `EncryptedFileSystem = OFF`.
2. Restart the management server with `--initial` or `--reload`.
3. Perform a rolling initial restart of the data nodes. Do *not* use any password-related options when restarting the node binaries.

   When restarted, each data node clears its on-disk state, and rebuilds it in unencrypted form.

To see whether file system encryption is properly configured, you can use a query against the `ndbinfo config_values` and `config_params` tables similar to this one:

```
mysql> SELECT v.node_id AS Node, p.param_name AS Parameter, v.config_value AS Value
    ->    FROM ndbinfo.config_values v
    ->  JOIN ndbinfo.config_params p
    ->    ON v.config_param=p.param_number
    ->  WHERE p.param_name='EncryptedFileSystem';
+------+----------------------+-------+
| Node | Parameter            | Value |
+------+----------------------+-------+
|    5 | EncryptedFileSystem  | 1     |
|    6 | EncryptedFileSystem  | 1     |
|    7 | EncryptedFileSystem  | 1     |
|    8 | EncryptedFileSystem  | 1     |
+------+----------------------+-------+
4 rows in set (0.10 sec)
```

Here, `EncryptedFileSystem` is equal to `1` on all data nodes, which means that filesystem encryption is enabled for this cluster.

### 23.6.14.2 NDB File System Encryption Implementation

For `NDB` Transparent Data Encryption (TDE), data nodes encrypt user data at rest, with security provided by a password (file system password), which is used to encrypt and decrypt a secrets file on

each data node. The secrets file contains a Node Master Key (NMK), a key used later to encrypt the different file types used for persistence. NDB TDE encrypts user data files including LCP files, redo log files, tablespace files, and undo log files.

You can use the ndbxfrm utility to see whether a file is encrypted, as shown here:

```
> ndbxfrm -i ndb_5_fs/LCP/0/T2F0.Data
File=ndb_5_fs/LCP/0/T2F0.Data, compression=no, encryption=yes
> ndbxfrm -i ndb_6_fs/LCP/0/T2F0.Data
File=ndb_6_fs/LCP/0/T2F0.Data, compression=no, encryption=no
```

Beginning with NDB 8.0.31, it is possible to obtain the key from the secrets file using the ndb_secretsfile_reader program added in that release, like this:

```
> ndb_secretsfile_reader --filesystem-password=54kl14 ndb_5_fs/D1/NDBCNTR/S0.sysfile
ndb_secretsfile_reader: [Warning] Using a password on the command line interface can be insecure.
cac256e18b2ddf6b5ef82d99a72f18e864b78453cc7fa40bfaf0c40b91122d18
```

The per-node key hierarchy can be represented as follows:

- A user-supplied passphrase (P) is processed by a key-derivation function using a random salt to generate a unique passphase key (PK).
- The PK (unique to each node) encrypts the data on each node in its own secrets file.
- The data in the secrets file includes a unique, randomly generated Node Master Key (NMK).
- The NMK encrypts (using wrapping) one or more randomly generated data encryption key (DEK) values in the header of each encrypted file (including LCP and TS files, and redo and undo logs).
- Data encryption key values ($DEK_0$, ..., $DEK_n$) are used for encryption of [subsets of] data in each file.

The passphrase indirectly encrypts the secrets file containing the random NMK, which encrypts a portion of the header of each encrypted file on the node. The encrypted file header contains random data keys used for the data in that file.

Encryption is implemented transparently by the NDBFS layer within the data nodes. NDBFS internal client blocks operate on their files as normal; NDBFS wraps the physical file with extra header and footer information supporting encryption, and encrypts and decrypts data as it is read from and written to the file. The wrapped file format is referred to as ndbxfrm1.

The node password is processed with PBKDF2 and the random salt to encrypt the secrets file, which contains the randomly generated NMK which is used to encrypt the randomly generated data encryption key in each encrypted file.

The work of encryption and decryption is performed in the NDBFS I/O threads (rather than in signal execution threads such as main, tc, ldm, or rep). This is similar to what happens with compressed LCPs and compressed backups, and normally results in increased I/O thread CPU usage; you may wish to adjust ThreadConfig (if in use) with regard to the I/O threads.

### 23.6.14.3 NDB File System Encryption Limitations

Transparent data encryption in NDB Cluster is subject to the following restrictions and limitations:

- The file system password must be supplied to each individual data node.
- File system password rotation requires an initial rolling restart of the data nodes; this must be performed manually, or by an application external to NDB).
- For a cluster with only a single replica (NoOfReplicas = 1), a full backup and restore is required for file system password rotation.
- Rotation of all data encryption keys requires an initial node restart.

## 23.6.15 NDB API Statistics Counters and Variables

A number of types of statistical counters relating to actions performed by or affecting `Ndb` objects are available. Such actions include starting and closing (or aborting) transactions; primary key and unique key operations; table, range, and pruned scans; threads blocked while waiting for the completion of various operations; and data and events sent and received by `NDBCLUSTER`. The counters are incremented inside the NDB kernel whenever NDB API calls are made or data is sent to or received by the data nodes. `mysqld` exposes these counters as system status variables; their values can be read in the output of `SHOW STATUS`, or by querying the Performance Schema `session_status` or `global_status` table. By comparing the values before and after statements operating on `NDB` tables, you can observe the corresponding actions taken on the API level, and thus the cost of performing the statement.

You can list all of these status variables using the following `SHOW STATUS` statement:

```
mysql> SHOW STATUS LIKE 'ndb_api%';
+-----------------------------------------------+-----------+
| Variable_name                                 | Value     |
+-----------------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count              | 297       |
| Ndb_api_wait_scan_result_count                | 0         |
| Ndb_api_wait_meta_request_count               | 321       |
| Ndb_api_wait_nanos_count                      | 228438645 |
| Ndb_api_bytes_sent_count                      | 33988     |
| Ndb_api_bytes_received_count                  | 66236     |
| Ndb_api_trans_start_count                     | 148       |
| Ndb_api_trans_commit_count                    | 148       |
| Ndb_api_trans_abort_count                     | 0         |
| Ndb_api_trans_close_count                     | 148       |
| Ndb_api_pk_op_count                           | 151       |
| Ndb_api_uk_op_count                           | 0         |
| Ndb_api_table_scan_count                      | 0         |
| Ndb_api_range_scan_count                      | 0         |
| Ndb_api_pruned_scan_count                     | 0         |
| Ndb_api_scan_batch_count                      | 0         |
| Ndb_api_read_row_count                        | 147       |
| Ndb_api_trans_local_read_row_count            | 37        |
| Ndb_api_adaptive_send_forced_count            | 3         |
| Ndb_api_adaptive_send_unforced_count          | 294       |
| Ndb_api_adaptive_send_deferred_count          | 0         |
| Ndb_api_event_data_count                      | 0         |
| Ndb_api_event_nondata_count                   | 0         |
| Ndb_api_event_bytes_count                     | 0         |
| Ndb_api_wait_exec_complete_count_slave        | 0         |
| Ndb_api_wait_scan_result_count_slave          | 0         |
| Ndb_api_wait_meta_request_count_slave         | 0         |
| Ndb_api_wait_nanos_count_slave                | 0         |
| Ndb_api_bytes_sent_count_slave                | 0         |
| Ndb_api_bytes_received_count_slave            | 0         |
| Ndb_api_trans_start_count_slave               | 0         |
| Ndb_api_trans_commit_count_slave              | 0         |
| Ndb_api_trans_abort_count_slave               | 0         |
| Ndb_api_trans_close_count_slave               | 0         |
| Ndb_api_pk_op_count_slave                     | 0         |
| Ndb_api_uk_op_count_slave                     | 0         |
| Ndb_api_table_scan_count_slave                | 0         |
| Ndb_api_range_scan_count_slave                | 0         |
| Ndb_api_pruned_scan_count_slave               | 0         |
| Ndb_api_scan_batch_count_slave                | 0         |
| Ndb_api_read_row_count_slave                  | 0         |
| Ndb_api_trans_local_read_row_count_slave      | 0         |
| Ndb_api_adaptive_send_forced_count_slave      | 0         |
| Ndb_api_adaptive_send_unforced_count_slave    | 0         |
| Ndb_api_adaptive_send_deferred_count_slave    | 0         |
| Ndb_api_wait_exec_complete_count_replica      | 0         |
| Ndb_api_wait_scan_result_count_replica        | 0         |
| Ndb_api_wait_meta_request_count_replica       | 0         |
| Ndb_api_wait_nanos_count_replica              | 0         |
| Ndb_api_bytes_sent_count_replica              | 0         |
| Ndb_api_bytes_received_count_replica          | 0         |
| Ndb_api_trans_start_count_replica             | 0         |
| Ndb_api_trans_commit_count_replica            | 0         |
```