MySQL Cluster NDB 8.0
Abstract

This is the MySQL Cluster NDB 8.0 extract from the MySQL 8.0 Reference Manual.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2021-02-26 (revision: 68830)
# Table of Contents

Preface and Legal Notices .................................................................................................................. vii
1 Preface and Notes .......................................................................................................................... 1
2 MySQL NDB Cluster 8.0 ................................................................................................................ 3
3 NDB Cluster Overview .................................................................................................................. 7
   3.1 NDB Cluster Core Concepts .................................................................................................. 8
   3.2 NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions ......................... 11
   3.3 NDB Cluster Hardware, Software, and Networking Requirements ............................... 14
   3.4 What is New in NDB Cluster ............................................................................................... 15
   3.5 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0 .......... 36
   3.6 MySQL Server Using InnoDB Compared with NDB Cluster ........................................... 40
      3.6.1 Differences Between the NDB and InnoDB Storage Engines .................................. 41
      3.6.2 NDB and InnoDB Workloads ....................................................................................... 42
      3.6.3 NDB and InnoDB Feature Usage Summary ............................................................... 43
   3.7 Known Limitations of NDB Cluster ..................................................................................... 43
      3.7.1 Noncompliance with SQL Syntax in NDB Cluster ..................................................... 44
      3.7.2 Limits and Differences of NDB Cluster from Standard MySQL Limits .................... 46
      3.7.3 Limits Relating to Transaction Handling in NDB Cluster ......................................... 47
      3.7.4 NDB Cluster Error Handling ...................................................................................... 50
      3.7.5 Limits Associated with Database Objects in NDB Cluster ......................................... 50
      3.7.6 Unsupported or Missing Features in NDB Cluster ....................................................... 50
      3.7.7 Limitations Relating to Performance in NDB Cluster ................................................ 51
      3.7.8 Issues Exclusive to NDB Cluster ................................................................................ 52
      3.7.9 Limitations Relating to NDB Cluster Disk Data Storage .......................................... 52
      3.7.10 Limitations Relating to Multiple NDB Cluster Nodes ............................................. 53
      3.7.11 Previous NDB Cluster Issues Resolved in NDB Cluster 8.0 .................................... 54
4 NDB Cluster Installation ................................................................................................................. 55
   4.1 Installation of NDB Cluster on Linux .................................................................................... 57
      4.1.1 Installing an NDB Cluster Binary Release on Linux .................................................. 58
      4.1.2 Installing NDB Cluster from RPM ............................................................................. 60
      4.1.3 Installing NDB Cluster Using .deb Files .................................................................. 64
      4.1.4 Building NDB Cluster from Source on Linux ............................................................. 64
   4.2 Installing NDB Cluster on Windows ..................................................................................... 65
      4.2.1 Installing NDB Cluster on Windows from a Binary Release .................................... 66
      4.2.2 Compiling and Installing NDB Cluster from Source on Windows ......................... 69
      4.2.3 Initial Startup of NDB Cluster on Windows ............................................................... 69
      4.2.4 Installing NDB Cluster Processes as Windows Services .......................................... 72
   4.3 Initial Configuration of NDB Cluster .................................................................................... 74
   4.4 Initial Setup of NDB Cluster .................................................................................................. 75
   4.5 NDB Cluster Example with Tables and Data ...................................................................... 76
   4.6 Safe Shutdown and Restart of NDB Cluster ....................................................................... 79
   4.7 Upgrading and Downgrading NDB Cluster ......................................................................... 80
   4.8 The NDB Cluster Auto-Installer (No longer supported) ..................................................... 83
      4.8.1 NDB Cluster Auto-Installer Requirements ................................................................. 83
      4.8.2 Using the NDB Cluster Auto-Installer ...................................................................... 84
5 Configuration of NDB Cluster ....................................................................................................... 105
   5.1 Quick Test Setup of NDB Cluster ......................................................................................... 105
   5.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables ............. 107
      5.2.1 NDB Cluster Data Node Configuration Parameters ................................................ 108
      5.2.2 NDB Cluster Management Node Configuration Parameters .................................. 108
      5.2.3 NDB Cluster SQL Node and API Node Configuration Parameters ....................... 109
      5.2.4 Other NDB Cluster Configuration Parameters ......................................................... 109
      5.2.5 NDB Cluster mysqld Option and Variable Reference ............................................. 109
   5.3 NDB Cluster Configuration Files ......................................................................................... 110
      5.3.1 NDB Cluster Configuration: Basic Example ............................................................... 111
      5.3.2 Recommended Starting Configuration for NDB Cluster .......................................... 113
MySQL Cluster NDB 8.0

5.3.3 NDB Cluster Connection Strings ................................................................. 116
5.3.4 Defining Computers in an NDB Cluster ......................................................... 117
5.3.5 Defining an NDB Cluster Management Server ................................................ 117
5.3.6 Defining NDB Cluster Data Nodes ................................................................. 122
5.3.7 Defining SQL and Other API Nodes in an NDB Cluster ................................. 177
5.3.8 Defining the System ..................................................................................... 182
5.3.9 MySQL Server Options and Variables for NDB Cluster ............................... 183
5.3.10 NDB Cluster TCP/IP Connections ................................................................ 237
5.3.11 NDB Cluster TCP/IP Connections Using Direct Connections ..................... 240
5.3.12 NDB Cluster Shared-Memory Connections .................................................... 241
5.3.13 Data Node Memory Management ................................................................ 244
5.3.14 Configuring NDB Cluster Send Buffer Parameters ................................. 248
5.4 Using High-Speed Interconnects with NDB Cluster ....................................... 248

6 NDB Cluster Programs ................................................................................... 251

6.1 ndbd — The NDB Cluster Data Node Daemon .................................................. 252
6.2 ndbinfo_select_all — Select From ndbinfo Tables .......................................... 259
6.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded) ................... 260
6.4 ndb_mgm — The NDB Cluster Management Server Daemon ........................... 261
6.5 ndb_mgm — The NDB Cluster Management Client .......................................... 269
6.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables ........................................................................................................... 272
6.7 ndb_config — Extract NDB Cluster Configuration Information ........................ 274
6.8 ndb_delete_all — Delete All Rows from an NDB Table ...................................... 283
6.9 ndb_desc — Describe NDB Tables .................................................................... 284
6.10 ndb_drop_index — Drop Index from an NDB Table ......................................... 289
6.11 ndb_drop_table — Drop an NDB Table ........................................................... 290
6.12 ndb_error_reporter — NDB Error-Reporting Utility ...................................... 291
6.13 ndb_import — Import CSV Data Into NDB ....................................................... 292
6.14 ndb_index_stat — NDB Index Statistics Utility .............................................. 304
6.15 ndb_move_data — NDB Data Copy Utility ....................................................... 309
6.16 ndb_perror — Obtain NDB Error Message Information ................................. 312
6.17 ndb_print_backup_file — Print NDB Backup File Contents ........................... 313
6.18 ndb_print_file — Print NDB Disk Data File Contents ...................................... 317
6.19 ndb_print_frag_file — Print NDB Fragment List File Contents ...................... 318
6.20 ndb_print_schema_file — Print NDB Schema File Contents ......................... 319
6.21 ndb_print_sys_file — Print NDB System File Contents .................................. 319
6.22 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log ........... 319
6.23 ndb_restore — Restore an NDB Cluster Backup .............................................. 322
6.23.1 Restoring an NDB Backup to a Different Version of NDB Cluster ............. 345
6.23.2 Restoring to a different number of data nodes .............................................. 346
6.23.3 Restoring from a backup taken in parallel .................................................... 349
6.24 ndb_select_all — Print Rows from an NDB Table .......................................... 350
6.25 ndb_select_count — Print Row Counts for NDB Tables ................................. 353
6.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED) ............................................................................................................. 354
6.27 ndb_show_tables — Display List of NDB Tables ............................................. 357
6.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator ......................... 359
6.29 ndb_top — View CPU usage information for NDB threads ............................ 361
6.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status ....................... 366
6.31 ndbxfm — Compress, Decompress, Encrypt, and Decrypt Files Created by NDB Cluster ............................................................................................................. 368
6.32 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs .................................................................................................. 371

7 Management of NDB Cluster .......................................................................... 377

7.1 Commands in the NDB Cluster Management Client ...................................... 379
7.2 NDB Cluster Log Messages ............................................................................ 384
7.2.1 NDB Cluster: Messages in the Cluster Log ................................................... 384
7.2.2 NDB Cluster Log Startup Messages ............................................................. 396
7.14 ndbinfo: The NDB Cluster Information Database ......................................................... 451
7.2.3 Event Buffer Reporting in the Cluster Log .......................................................... 396
7.2.4 NDB Cluster: NDB Transporter Errors ............................................................... 397
7.3 Event Reports Generated in NDB Cluster ............................................................... 399
7.3.1 NDB Cluster Logging Management Commands .................................................... 400
7.3.2 NDB Cluster Log Events ....................................................................................... 402
7.3.3 Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client .......... 407
7.4 Summary of NDB Cluster Start Phases ................................................................... 409
7.5 Performing a Rolling Restart of an NDB Cluster ....................................................... 411
7.6 NDB Cluster Single User Mode ................................................................................. 413
7.7 Adding NDB Cluster Data Nodes Online ................................................................. 414
7.7.1 Adding NDB Cluster Data Nodes Online: General Issues ........................................ 414
7.7.2 Adding NDB Cluster Data Nodes Online: Basic procedure .................................... 416
7.7.3 Adding NDB Cluster Data Nodes Online: Detailed Example .................................. 417
7.8 Online Backup of NDB Cluster ............................................................................. 424
7.8.1 NDB Cluster Backup Concepts ........................................................................... 424
7.8.2 Using The NDB Cluster Management Client to Create a Backup ......................... 425
7.8.3 Configuration for NDB Cluster Backups ............................................................... 428
7.8.4 NDB Cluster Backup Troubleshooting .................................................................. 429
7.8.5 Taking an NDB Backup with Parallel Data Nodes ............................................... 429
7.9 MySQL Server Usage for NDB Cluster .................................................................. 430
7.10 NDB Cluster Disk Data Tables ............................................................................. 431
7.10.1 NDB Cluster Disk Data Objects ........................................................................ 431
7.10.2 NDB Cluster Disk Data Storage Requirements ................................................... 436
7.11 Online Operations with ALTER TABLE in NDB Cluster ......................................... 437
7.12 Distributed MySQL Privileges with NDB_STORED_USER ....................................... 440
7.13 NDB API Statistics Counters and Variables ........................................................... 441
7.14 ndbinfo: The NDB Cluster Information Database ......................................................... 451
7.14.1 The ndbinfo arbitrator_validity_detail Table ....................................................... 455
7.14.2 The ndbinfo arbitrator_validity_summary Table .................................................. 456
7.14.3 The ndbinfo backup_id Table .............................................................................. 456
7.14.4 The ndbinfo blocks Table .................................................................................... 457
7.14.5 The ndbinfo cluster_locks Table ......................................................................... 458
7.14.6 The ndbinfo cluster_operations Table ............................................................... 459
7.14.7 The ndbinfo cluster_transactions Table ............................................................. 460
7.14.8 The ndbinfo config_nodes Table ....................................................................... 462
7.14.9 The ndbinfo config_params Table ...................................................................... 462
7.14.10 The ndbinfo config_values Table ..................................................................... 463
7.14.11 The ndbinfo counters Table .............................................................................. 465
7.14.12 The ndbinfo cpudata Table ............................................................................... 467
7.14.13 The ndbinfo cpudata_1sec Table ...................................................................... 468
7.14.14 The ndbinfo cpudata_20sec Table .................................................................... 468
7.14.15 The ndbinfo cpudata_50ms Table .................................................................... 469
7.14.16 The ndbinfo cpuninfo Table .............................................................................. 470
7.14.17 The ndbinfo cpustat Table ............................................................................... 471
7.14.18 The ndbinfo cpustat_50ms Table ...................................................................... 471
7.14.19 The ndbinfo cpustat_1sec Table ...................................................................... 472
7.14.20 The ndbinfo cpustat_20sec Table .................................................................... 473
7.14.21 The ndbinfo dict_obj_info Table ..................................................................... 474
7.14.22 The ndbinfo dict_obj_tree Table ..................................................................... 475
7.14.23 The ndbinfo dict_obj_types Table .................................................................... 478
7.14.24 The ndbinfo disk_write_speed_base Table ......................................................... 478
7.14.25 The ndbinfo disk_write_speed_aggregate Table ................................................ 478
7.14.26 The ndbinfo disk_write_speed_aggregate_node Table ....................................... 480
7.14.27 The ndbinfo diskpagebuffer Table ................................................................... 480
7.14.28 The ndbinfo diskstat Table ............................................................................... 482
7.14.29 The ndbinfo diskstats_1sec Table .................................................................... 483
7.14.30 The ndbinfo error_messages Table .................................................................. 484
7.14.31 The ndbinfo hwinfo Table ............................................................................... 485
Preface and Legal Notices

This is the MySQL Cluster NDB 8.0 extract from the MySQL 8.0 Reference Manual.

**Licensing information—MySQL NDB Cluster 8.0.** If you are using a *Community* release of MySQL NDB Cluster 8.0, see the MySQL NDB Cluster 8.0 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

**Legal Notices**

Copyright © 1997, 2021, Oracle and/or its affiliates.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

**U.S. GOVERNMENT END USERS:** Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs) and Oracle computer documentation or other Oracle data delivered to or accessed by U.S. Government end users are "commercial computer software" or "commercial computer software documentation" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, the use, reproduction, duplication, release, display, disclosure, modification, preparation of derivative works, and/or adaptation of i) Oracle programs (including any operating system, integrated software, any programs embedded, installed or activated on delivered hardware, and modifications of such programs), ii) Oracle computer documentation and/or iii) other Oracle data, is subject to the rights and limitations specified in the license contained in the applicable contract. The terms governing the U.S. Government’s use of Oracle cloud services are defined by the applicable contract for such services. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Inside are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Epyc, and the AMD logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to
your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

**Documentation Accessibility**

For information about Oracle's commitment to accessibility, visit the Oracle Accessibility Program website at https://www.oracle.com/corporate/accessibility/.

**Access to Oracle Support for Accessibility**

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
Chapter 1 Preface and Notes

This is an extract from the Reference Manual for the MySQL Database System, version 8.0. It contains information about MySQL NDB Cluster 8.0 releases through MySQL NDB Cluster 8.0.24. Differences between minor versions of NDB Cluster covered in this extract are noted in the present text with reference to the NDBCLUSTER storage version number (8.0.x); which is the same as the version number of the MySQL Server 8.0 on which the NDB Cluster release is based, are noted with reference to MySQL 8.0 releases (8.0.x). For license information, see the legal notice.

This extract is not intended for use with older versions of the NDB Cluster software due to the many functional and other differences between current and previous versions. For information about previous NDB Cluster versions, see MySQL NDB Cluster 7.6. This extract provides information that is specific to NDB Cluster and is not intended to replace the MySQL 8.0 Reference Manual, which provides additional information about MySQL 8.0 which may also be necessary for use of MySQL NDB Cluster 8.0. If you are using MySQL Server 5.7 or an earlier release of the MySQL software, please refer to the appropriate manual. For example, MySQL 5.7 Reference Manual, covers the 5.7 series of MySQL Server releases.
Chapter 2 MySQL NDB Cluster 8.0

MySQL NDB Cluster is a high-availability, high-redundancy version of MySQL adapted for the distributed computing environment. The most recent NDB Cluster release series uses version 8 of the NDB storage engine (also known as NDBCLUSTER) to enable running several computers with MySQL servers and other software in a cluster. NDB Cluster 8.0, now available as a General Availability (GA) release beginning with version 8.0.19, incorporates version 8.0 of the NDB storage engine. NDB Cluster 7.6 and NDB Cluster 7.5, still available as GA releases, use versions 7.6 and 7.5 of NDB, respectively. Previous GA releases still available for use in production, NDB Cluster 7.4 and NDB Cluster 7.3, incorporate NDB versions 7.4 and 7.3, respectively. NDB 7.2 and older release series are no longer supported or maintained.

Support for the NDB storage engine is not included in standard MySQL Server 8.0 binaries built by Oracle. Instead, users of NDB Cluster binaries from Oracle should upgrade to the most recent binary release of NDB Cluster for supported platforms—these include RPMs that should work with most Linux distributions. NDB Cluster 8.0 users who build from source should use the sources provided for MySQL 8.0 and build with the options required to provide NDB support. (Locations where the sources can be obtained are listed later in this section.)

Important

MySQL NDB Cluster does not support InnoDB Cluster, which must be deployed using MySQL Server 8.0 with the InnoDB storage engine as well as additional applications that are not included in the NDB Cluster distribution. MySQL Server 8.0 binaries cannot be used with MySQL NDB Cluster. For more information about deploying and using InnoDB Cluster, see Using MySQL AdminAPI, Section 3.6, “MySQL Server Using InnoDB Compared with NDB Cluster”, discusses differences between the NDB and InnoDB storage engines.

This chapter contains information about NDB Cluster 8.0 releases through 8.0.24. NDB Cluster 8.0 is now available (beginning with NDB 8.0.19) as a General Availability release, and recommended for new deployments; the latest available release is NDB 8.0.23. NDB Cluster 7.6 and 7.5 are previous GA releases still supported in production; for information about NDB Cluster 7.6, see What is New in NDB Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB Cluster 7.5. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production, although we recommend that new deployments for production use NDB Cluster 8.0; see MySQL NDB Cluster 7.3 and NDB Cluster 7.4.

Supported Platforms. NDB Cluster is currently available and supported on a number of platforms. For exact levels of support available for on specific combinations of operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

Availability. NDB Cluster binary and source packages are available for supported platforms from https://dev.mysql.com/downloads/cluster/.

NDB Cluster release numbers. NDB 8.0 follows the same release pattern as the MySQL Server 8.0 series of releases, beginning with MySQL 8.0.13 and MySQL NDB Cluster 8.0.13. In this Manual and other MySQL documentation, we identify these and later NDB Cluster releases employing a version number that begins with “NDB”. This version number is that of the NDBCLUSTER storage engine used in the NDB 8.0 release, and is the same as the MySQL 8.0 server version on which the NDB Cluster 8.0 release is based.

Version strings used in NDB Cluster software. The version string displayed by the mysql client supplied with the MySQL NDB Cluster distribution uses this format:

```
mysql-<mysql_server_version>-cluster
```

mysql_server_version represents the version of the MySQL Server on which the NDB Cluster release is based. For all NDB Cluster 8.0 releases, this is 8.0.n, where n is the release number.
Building from source using `-DWITH_NDBCLUSTER` or the equivalent adds the `-cluster` suffix to the version string. (See Section 4.1.4, “Building NDB Cluster from Source on Linux”, and Section 4.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.) You can see this format used in the `mysql` client, as shown here:

```
shell> mysql
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 8.0.24-cluster Source distribution
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql> SELECT VERSION()\G
******** ******************* 1. row *******************
VERSION(): 8.0.24-cluster
1 row in set (0.00 sec)
```

The first General Availability release of NDB Cluster using MySQL 8.0 is NDB 8.0.19, using MySQL 8.0.19.

The version string displayed by other NDB Cluster programs not normally included with the MySQL 8.0 distribution uses this format:

```
mysql=mysql_server_version ndb=ndb_engine_version
```

`mysql_server_version` represents the version of the MySQL Server on which the NDB Cluster release is based. For all NDB Cluster 8.0 releases, this is `8.0.0`, where `0` is the release number. `ndb_engine_version` is the version of the NDB storage engine used by this release of the NDB Cluster software. For all NDB 8.0 releases, this number is the same as the MySQL Server version. You can see this format used in the output of the `SHOW` command in the `ndb_mgm` client, like this:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------
[ndbd(NDB)] 2 node(s)
id=1 @10.0.10.6 (mysql-8.0.25 ndb-8.0.24, Nodegroup: 0, *)
id=2 @10.0.10.8 (mysql-8.0.25 ndb-8.0.24, Nodegroup: 0)
[ndbd_mgmd(MGM)] 1 node(s)
id=3 @10.0.10.2 (mysql-8.0.25 ndb-8.0.24)
[mysqld(API)] 2 node(s)
id=4 @10.0.10.10 (mysql-8.0.25 ndb-8.0.24)
id=5 (not connected, accepting connect from any host)
```

**Compatibility with standard MySQL 8.0 releases.** While many standard MySQL schemas and applications can work using NDB Cluster, it is also true that unmodified applications and database schemas may be slightly incompatible or have suboptimal performance when run using NDB Cluster (see Section 3.7, “Known Limitations of NDB Cluster”). Most of these issues can be overcome, but this also means that you are very unlikely to be able to switch an existing application datastore —that currently uses, for example, MyISAM or InnoDB—to use the NDB storage engine without allowing for the possibility of changes in schemas, queries, and applications. A `mysqld` compiled without NDB support (that is, built without `-DWITH_NDBCLUSTER_STORAGE_ENGINE` or its alias `-DWITH_NDBCLUSTER`) cannot function as a drop-in replacement for a `mysqld` that is built with it.

**NDB Cluster development source trees.** NDB Cluster development trees can also be accessed from [https://github.com/mysql/mysql-server](https://github.com/mysql/mysql-server).

The NDB Cluster development sources maintained at [https://github.com/mysql/mysql-server](https://github.com/mysql/mysql-server) are licensed under the GPL. For information about obtaining MySQL sources using Git and building them yourself, see Installing MySQL Using a Development Source Tree.

**Note**

As with MySQL Server 8.0, NDB Cluster 8.0 releases are built using **CMake**.

NDB Cluster 8.0 is available beginning with NDB 8.0.19 as a General Availability release, and is recommended for new deployments. NDB Cluster 7.6 and 7.5 are previous GA releases still supported
in production; for information about NDB Cluster 7.6, see What is New in NDB Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB Cluster 7.5. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production, although we recommend that new deployments for production use NDB Cluster 8.0; see MySQL NDB Cluster 7.3 and NDB Cluster 7.4.

The contents of this chapter are subject to revision as NDB Cluster continues to evolve. Additional information regarding NDB Cluster can be found on the MySQL website at http://www.mysql.com/products/cluster/.

**Additional Resources.** More information about NDB Cluster can be found in the following places:

- For answers to some commonly asked questions about NDB Cluster, see Appendix A, *MySQL 8.0 FAQ: NDB Cluster*.
- Many NDB Cluster users and developers blog about their experiences with NDB Cluster, and make feeds of these available through PlanetMySQL.
NDB Cluster is a technology that enables clustering of in-memory databases in a shared-nothing system. The shared-nothing architecture enables the system to work with very inexpensive hardware, and with a minimum of specific requirements for hardware or software.

NDB Cluster is designed not to have any single point of failure. In a shared-nothing system, each component is expected to have its own memory and disk, and the use of shared storage mechanisms such as network shares, network file systems, and SANs is not recommended or supported.

NDB Cluster integrates the standard MySQL server with an in-memory clustered storage engine called NDB (which stands for “Network DataBase”). In our documentation, the term NDB refers to the part of the setup that is specific to the storage engine, whereas “MySQL NDB Cluster” refers to the combination of one or more MySQL servers with the NDB storage engine.

An NDB Cluster consists of a set of computers, known as hosts, each running one or more processes. These processes, known as nodes, may include MySQL servers (for access to NDB data), data nodes (for storage of the data), one or more management servers, and possibly other specialized data access programs. The relationship of these components in an NDB Cluster is shown here:
All these programs work together to form an NDB Cluster (see Chapter 6, *NDB Cluster Programs*). When data is stored by the NDB storage engine, the tables (and table data) are stored in the data nodes. Such tables are directly accessible from all other MySQL servers (SQL nodes) in the cluster. Thus, in a payroll application storing data in a cluster, if one application updates the salary of an employee, all other MySQL servers that query this data can see this change immediately.

Although an NDB Cluster SQL node uses the `mysqld` server daemon, it differs in a number of critical respects from the `mysqld` binary supplied with the MySQL 8.0 distributions, and the two versions of `mysqld` are not interchangeable.

In addition, a MySQL server that is not connected to an NDB Cluster cannot use the NDB storage engine and cannot access any NDB Cluster data.

The data stored in the data nodes for NDB Cluster can be mirrored; the cluster can handle failures of individual data nodes with no other impact than that a small number of transactions are aborted due to losing the transaction state. Because transactional applications are expected to handle transaction failure, this should not be a source of problems.

Individual nodes can be stopped and restarted, and can then rejoin the system (cluster). Rolling restarts (in which all nodes are restarted in turn) are used in making configuration changes and software upgrades (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”). Rolling restarts are also used as part of the process of adding new data nodes online (see Section 7.7, “Adding NDB Cluster Data Nodes Online”). For more information about data nodes, how they are organized in an NDB Cluster, and how they handle and store NDB Cluster data, see Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”.

Backing up and restoring NDB Cluster databases can be done using the NDB-native functionality found in the NDB Cluster management client and the `ndb_restore` program included in the NDB Cluster distribution. For more information, see Section 7.8, “Online Backup of NDB Cluster”, and Section 6.23, “ndb_restore — Restore an NDB Cluster Backup”. You can also use the standard MySQL functionality provided for this purpose in `mysqldump` and the MySQL server. See `mysqldump — A Database Backup Program`, for more information.

NDB Cluster nodes can employ different transport mechanisms for inter-node communications; TCP/IP over standard 100 Mbps or faster Ethernet hardware is used in most real-world deployments.

### 3.1 NDB Cluster Core Concepts

**NDBCluster** (also known as **NDB**) is an in-memory storage engine offering high-availability and data-persistence features.
The NDB Cluster storage engine can be configured with a range of failover and load-balancing options, but it is easiest to start with the storage engine at the cluster level. NDB Cluster's NDB storage engine contains a complete set of data, dependent only on other data within the cluster itself.

The “Cluster” portion of NDB Cluster is configured independently of the MySQL servers. In an NDB Cluster, each part of the cluster is considered to be a node.

Note

In many contexts, the term “node” is used to indicate a computer, but when discussing NDB Cluster it means a process. It is possible to run multiple nodes on a single computer; for a computer on which one or more cluster nodes are being run we use the term cluster host.

There are three types of cluster nodes, and in a minimal NDB Cluster configuration, there are at least three nodes, one of each of these types:

- **Management node**: The role of this type of node is to manage the other nodes within the NDB Cluster, performing such functions as providing configuration data, starting and stopping nodes, and running backups. Because this node type manages the configuration of the other nodes, a node of this type should be started first, before any other node. A management node is started with the command ndb_mgmd.

- **Data node**: This type of node stores cluster data. There are as many data nodes as there are fragment replicas, times the number of fragments (see Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”). For example, with two fragment replicas, each having two fragments, you need four data nodes. One fragment replica is sufficient for data storage, but provides no redundancy; therefore, it is recommended to have two (or more) fragment replicas to provide redundancy, and thus high availability. A data node is started with the command ndbd (see Section 6.1, “ndbd — The NDB Cluster Data Node Daemon”) or ndbmtd (see Section 6.3, “ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)”).

NDB Cluster tables are normally stored completely in memory rather than on disk (this is why we refer to NDB Cluster as an in-memory database). However, some NDB Cluster data can be stored on disk; see Section 7.10, “NDB Cluster Disk Data Tables”, for more information.

- **SQL node**: This is a node that accesses the cluster data. In the case of NDB Cluster, an SQL node is a traditional MySQL server that uses the NDBCluster storage engine. An SQL node is a mysqld process started with the --ndbcluster and --ndb-connectstring options, which are explained elsewhere in this chapter, possibly with additional MySQL server options as well.

An SQL node is actually just a specialized type of API node, which designates any application which accesses NDB Cluster data. Another example of an API node is the ndb_restore utility that is used to restore a cluster backup. It is possible to write such applications using the NDB API. For basic information about the NDB API, see Getting Started with the NDB API.

Important

It is not realistic to expect to employ a three-node setup in a production environment. Such a configuration provides no redundancy; to benefit from NDB Cluster’s high-availability features, you must use multiple data and SQL nodes. The use of multiple management nodes is also highly recommended.

For a brief introduction to the relationships between nodes, node groups, fragment replicas, and partitions in NDB Cluster, see Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”.

Configuration of a cluster involves configuring each individual node in the cluster and setting up individual communication links between nodes. NDB Cluster is currently designed with the intention that data nodes are homogeneous in terms of processor power, memory space, and bandwidth. In
addition, to provide a single point of configuration, all configuration data for the cluster as a whole is located in one configuration file.

The management server manages the cluster configuration file and the cluster log. Each node in the cluster retrieves the configuration data from the management server, and so requires a way to determine where the management server resides. When interesting events occur in the data nodes, the nodes transfer information about these events to the management server, which then writes the information to the cluster log.

In addition, there can be any number of cluster client processes or applications. These include standard MySQL clients, NDB-specific API programs, and management clients. These are described in the next few paragraphs.

**Standard MySQL clients.** NDB Cluster can be used with existing MySQL applications written in PHP, Perl, C, C++, Java, Python, Ruby, and so on. Such client applications send SQL statements to and receive responses from MySQL servers acting as NDB Cluster SQL nodes in much the same way that they interact with standalone MySQL servers.

MySQL clients using an NDB Cluster as a data source can be modified to take advantage of the ability to connect with multiple MySQL servers to achieve load balancing and failover. For example, Java clients using Connector/J 5.0.6 and later can use `jdbc:mysql:loadbalance://` URLs (improved in Connector/J 5.1.7) to achieve load balancing transparently; for more information about using Connector/J with NDB Cluster, see Using Connector/J with NDB Cluster.

**NDB client programs.** Client programs can be written that access NDB Cluster data directly from the NDBCLUSTER storage engine, bypassing any MySQL Servers that may be connected to the cluster, using the NDB API, a high-level C++ API. Such applications may be useful for specialized purposes where an SQL interface to the data is not needed. For more information, see The NDB API.

NDB-specific Java applications can also be written for NDB Cluster using the NDB Cluster Connector for Java. This NDB Cluster Connector includes ClusterJ, a high-level database API similar to object-relational mapping persistence frameworks such as Hibernate and JPA that connect directly to NDBCLUSTER, and so does not require access to a MySQL Server. See Java and NDB Cluster, and The ClusterJ API and Data Object Model, for more information.

NDB Cluster also supports applications written in JavaScript using Node.js. The MySQL Connector for JavaScript includes adapters for direct access to the NDB storage engine and as well as for the MySQL Server. Applications using this Connector are typically event-driven and use a domain object model similar in many ways to that employed by ClusterJ. For more information, see MySQL NoSQL Connector for JavaScript.

**Management clients.** These clients connect to the management server and provide commands for starting and stopping nodes gracefully, starting and stopping message tracing (debug versions only), showing node versions and status, starting and stopping backups, and so on. An example of this type of program is the `ndb_mgm` management client supplied with NDB Cluster (see Section 6.5, “ndb_mgm — The NDB Cluster Management Client”). Such applications can be written using the MGM API, a C-language API that communicates directly with one or more NDB Cluster management servers. For more information, see The MGM API.

Oracle also makes available MySQL Cluster Manager, which provides an advanced command-line interface simplifying many complex NDB Cluster management tasks, such as 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 1.4.8 provides experimental support for NDB 8.0. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

**Event logs.** NDB Cluster logs events by category (startup, shutdown, errors, checkpoints, and so on), priority, and severity. A complete listing of all reportable events may be found in Section 7.3, “Event Reports Generated in NDB Cluster”. Event logs are of the two types listed here:

- **Cluster log:** Keeps a record of all desired reportable events for the cluster as a whole.
• **Node log**: A separate log which is also kept for each individual node.

**Note**
Under normal circumstances, it is necessary and sufficient to keep and examine only the cluster log. The node logs need be consulted only for application development and debugging purposes.

**Checkpoint.** Generally speaking, when data is saved to disk, it is said that a *checkpoint* has been reached. More specific to NDB Cluster, a checkpoint is a point in time where all committed transactions are stored on disk. With regard to the NDB storage engine, there are two types of checkpoints which work together to ensure that a consistent view of the cluster’s data is maintained. These are shown in the following list:

- **Local Checkpoint (LCP)**: This is a checkpoint that is specific to a single node; however, LCPs take place for all nodes in the cluster more or less concurrently. An LCP usually occurs every few minutes; the precise interval varies, and depends upon the amount of data stored by the node, the level of cluster activity, and other factors.

  NDB 8.0 supports partial LCPs, which can significantly improve performance under some conditions. See the descriptions of the `EnablePartialLcp` and `RecoveryWork` configuration parameters which enable partial LCPs and control the amount of storage they use.

- **Global Checkpoint (GCP)**: A GCP occurs every few seconds, when transactions for all nodes are synchronized and the redo-log is flushed to disk.

For more information about the files and directories created by local checkpoints and global checkpoints, see NDB Cluster Data Node File System Directory.

### 3.2 NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions

This section discusses the manner in which NDB Cluster divides and duplicates data for storage.

A number of concepts central to an understanding of this topic are discussed in the next few paragraphs.

**Data node.** An ndbd or ndbmtd process, which stores one or more fragment replicas—that is, copies of the partitions (discussed later in this section) assigned to the node group of which the node is a member.

Each data node should be located on a separate computer. While it is also possible to host multiple data node processes on a single computer, such a configuration is not usually recommended.

It is common for the terms “node” and “data node” to be used interchangeably when referring to an ndbd or ndbmtd process; where mentioned, management nodes (ndb_mgmd processes) and SQL nodes (mysqld processes) are specified as such in this discussion.

**Node group.** A node group consists of one or more nodes, and stores partitions, or sets of fragment replicas (see next item).

The number of node groups in an NDB Cluster is not directly configurable; it is a function of the number of data nodes and of the number of fragment replicas (`NoOfReplicas` configuration parameter), as shown here:

\[
[# \text{ of node groups}] = \frac{[# \text{ of data nodes}]}{\text{NoOfReplicas}}
\]

Thus, an NDB Cluster with 4 data nodes has 4 node groups if `NoOfReplicas` is set to 1 in the `config.ini` file, 2 node groups if `NoOfReplicas` is set to 2, and 1 node group if `NoOfReplicas` is set to 4. Fragment replicas are discussed later in this section; for more information about `NoOfReplicas`, see Section 5.3.6, “Defining NDB Cluster Data Nodes”.
NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions

Note

All node groups in an NDB Cluster must have the same number of data nodes.

You can add new node groups (and thus new data nodes) online, to a running NDB Cluster; see Section 7.7, “Adding NDB Cluster Data Nodes Online”, for more information.

Partition.  This is a portion of the data stored by the cluster. Each node is responsible for keeping at least one copy of any partitions assigned to it (that is, at least one fragment replica) available to the cluster.

The number of partitions used by default by NDB Cluster depends on the number of data nodes and the number of LDM threads in use by the data nodes, as shown here:

\[
\text{[# of partitions]} = \text{[# of data nodes]} \times \text{[# of LDM threads]}
\]

When using data nodes running ndbmt, the number of LDM threads is controlled by the setting for MaxNoOfExecutionThreads. When using ndbd there is a single LDM thread, which means that there are as many cluster partitions as nodes participating in the cluster. This is also the case when using ndbmt with MaxNoOfExecutionThreads set to 3 or less. (You should be aware that the number of LDM threads increases with the value of this parameter, but not in a strictly linear fashion, and that there are additional constraints on setting it; see the description of MaxNoOfExecutionThreads for more information.)

NDB and user-defined partitioning.  NDB Cluster normally partitions NDBCLUSTER tables automatically. However, it is also possible to employ user-defined partitioning with NDBCLUSTER tables. This is subject to the following limitations:

1. Only the KEY and LINEAR KEY partitioning schemes are supported in production with NDB tables.

2. The maximum number of partitions that may be defined explicitly for any NDB table is 8 * [number of LDM threads] * [number of node groups], the number of node groups in an NDB Cluster being determined as discussed previously in this section. When running ndbd for data node processes, setting the number of LDM threads has no effect (since ThreadConfig applies only to ndbmt); in such cases, this value can be treated as though it were equal to 1 for purposes of performing this calculation.

See Section 6.3, “ndbmt — The NDB Cluster Data Node Daemon (Multi-Threaded)”, for more information.

For more information relating to NDB Cluster and user-defined partitioning, see Section 3.7, “Known Limitations of NDB Cluster”, and Partitioning Limitations Relating to Storage Engines.

Fragment replica.  This is a copy of a cluster partition. Each node in a node group stores a fragment replica. Also sometimes known as a partition replica. The number of fragment replicas is equal to the number of nodes per node group.

A fragment replica belongs entirely to a single node; a node can (and usually does) store several fragment replicas.

The following diagram illustrates an NDB Cluster with four data nodes running ndbd, arranged in two node groups of two nodes each; nodes 1 and 2 belong to node group 0, and nodes 3 and 4 belong to node group 1.

Note

Only data nodes are shown here; although a working NDB Cluster requires an ndb_mgmd process for cluster management and at least one SQL node to access the data stored by the cluster, these have been omitted from the figure for clarity.
The data stored by the cluster is divided into four partitions, numbered 0, 1, 2, and 3. Each partition is stored—in multiple copies—on the same node group. Partitions are stored on alternate node groups as follows:

- Partition 0 is stored on node group 0; a primary fragment replica (primary copy) is stored on node 1, and a backup fragment replica (backup copy of the partition) is stored on node 2.
- Partition 1 is stored on the other node group (node group 1); this partition's primary fragment replica is on node 3, and its backup fragment replica is on node 4.
- Partition 2 is stored on node group 0. However, the placing of its two fragment replicas is reversed from that of Partition 0; for Partition 2, the primary fragment replica is stored on node 2, and the backup on node 1.
- Partition 3 is stored on node group 1, and the placement of its two fragment replicas are reversed from those of partition 1. That is, its primary fragment replica is located on node 4, with the backup on node 3.

What this means regarding the continued operation of an NDB Cluster is this: so long as each node group participating in the cluster has at least one node operating, the cluster has a complete copy of all data and remains viable. This is illustrated in the next diagram.
In this example, the cluster consists of two node groups each consisting of two data nodes. Each data node is running an instance of `ndbd`. Any combination of at least one node from node group 0 and at least one node from node group 1 is sufficient to keep the cluster “alive”. However, if both nodes from a single node group fail, the combination consisting of the remaining two nodes in the other node group is not sufficient. In this situation, the cluster has lost an entire partition and so can no longer provide access to a complete set of all NDB Cluster data.

The maximum number of node groups supported for a single NDB Cluster instance is 48.

### 3.3 NDB Cluster Hardware, Software, and Networking Requirements

One of the strengths of NDB Cluster is that it can be run on commodity hardware and has no unusual requirements in this regard, other than for large amounts of RAM, due to the fact that all live data storage is done in memory. (It is possible to reduce this requirement using Disk Data tables—see Section 7.10, “NDB Cluster Disk Data Tables”, for more information about these.) Naturally, multiple and faster CPUs can enhance performance. Memory requirements for other NDB Cluster processes are relatively small.

The software requirements for NDB Cluster are also modest. Host operating systems do not require any unusual modules, services, applications, or configuration to support NDB Cluster. For supported operating systems, a standard installation should be sufficient. The MySQL software requirements are simple: all that is needed is a production release of NDB Cluster. It is not strictly necessary to compile MySQL yourself merely to be able to use NDB Cluster. We assume that you are using the binaries appropriate to your platform, available from the NDB Cluster software downloads page at https://dev.mysql.com/downloads/cluster/.

For communication between nodes, NDB Cluster supports TCP/IP networking in any standard topology, and the minimum expected for each host is a standard 100 Mbps Ethernet card, plus a switch, hub, or router to provide network connectivity for the cluster as a whole. We strongly recommend that an NDB Cluster be run on its own subnet which is not shared with machines not forming part of the cluster for the following reasons:

- **Security.** Communications between NDB Cluster nodes are not encrypted or shielded in any way. The only means of protecting transmissions within an NDB Cluster is to run your NDB Cluster on a protected network. If you intend to use NDB Cluster for Web applications, the cluster should
definitely reside behind your firewall and not in your network's De-Militarized Zone (DMZ) or elsewhere.


• **Efficiency.** Setting up an NDB Cluster on a private or protected network enables the cluster to make exclusive use of bandwidth between cluster hosts. Using a separate switch for your NDB Cluster not only helps protect against unauthorized access to NDB Cluster data, it also ensures that NDB Cluster nodes are shielded from interference caused by transmissions between other computers on the network. For enhanced reliability, you can use dual switches and dual cards to remove the network as a single point of failure; many device drivers support failover for such communication links.

**Network communication and latency.** NDB Cluster requires communication between data nodes and API nodes (including SQL nodes), as well as between data nodes and other data nodes, to execute queries and updates. Communication latency between these processes can directly affect the observed performance and latency of user queries. In addition, to maintain consistency and service despite the silent failure of nodes, NDB Cluster uses heartbeating and timeout mechanisms which treat an extended loss of communication from a node as node failure. This can lead to reduced redundancy. Recall that, to maintain data consistency, an NDB Cluster shuts down when the last node in a node group fails. Thus, to avoid increasing the risk of a forced shutdown, breaks in communication between nodes should be avoided wherever possible.

The failure of a data or API node results in the abort of all uncommitted transactions involving the failed node. Data node recovery requires synchronization of the failed node’s data from a surviving data node, and re-establishment of disk-based redo and checkpoint logs, before the data node returns to service. This recovery can take some time, during which the Cluster operates with reduced redundancy.

Heartbeating relies on timely generation of heartbeat signals by all nodes. This may not be possible if the node is overloaded, has insufficient machine CPU due to sharing with other programs, or is experiencing delays due to swapping. If heartbeat generation is sufficiently delayed, other nodes treat the node that is slow to respond as failed.

This treatment of a slow node as a failed one may or may not be desirable in some circumstances, depending on the impact of the node’s slowed operation on the rest of the cluster. When setting timeout values such as `HeartbeatIntervalDbDb` and `HeartbeatIntervalDbApi` for NDB Cluster, care must be taken to achieve quick detection, failover, and return to service, while avoiding potentially expensive false positives.

Where communication latencies between data nodes are expected to be higher than would be expected in a LAN environment (on the order of 100 µs), timeout parameters must be increased to ensure that any allowed periods of latency periods are well within configured timeouts. Increasing timeouts in this way has a corresponding effect on the worst-case time to detect failure and therefore time to service recovery.

LAN environments can typically be configured with stable low latency, and such that they can provide redundancy with fast failover. Individual link failures can be recovered from with minimal and controlled latency visible at the TCP level (where NDB Cluster normally operates). WAN environments may offer a range of latencies, as well as redundancy with slower failover times. Individual link failures may require route changes to propagate before end-to-end connectivity is restored. At the TCP level this can appear as large latencies on individual channels. The worst-case observed TCP latency in these scenarios is related to the worst-case time for the IP layer to reroute around the failures.

### 3.4 What is New in NDB Cluster

The following sections describe changes in the implementation of NDB Cluster in MySQL NDB Cluster 8.0 through 8.0.24, as compared to earlier release series. NDB Cluster 8.0 is available as a General Availability (GA) release, beginning with NDB 8.0.19. NDB Cluster 7.6 and 7.5 are previous GA releases still supported in production; for information about NDB Cluster 7.6, see What is New in NDB
What is New in NDB Cluster 8.0

Cluster 7.6. For similar information about NDB Cluster 7.5, see What is New in NDB Cluster 7.5. NDB Cluster 7.4 and 7.3 are previous GA releases still supported in production, although we recommend that new deployments for production use NDB Cluster 8.0; see MySQL NDB Cluster 7.3 and NDB Cluster 7.4.

What is New in NDB Cluster 8.0

Major changes and new features in NDB Cluster 8.0 which are likely to be of interest are shown in the following list:

- **Compatibility enhancements.** The following changes reduce longstanding nonessential differences in NDB behavior as compared to that of other MySQL storage engines:
  - **Development in parallel with MySQL server.** Beginning with this release, MySQL NDB Cluster is being developed in parallel with the standard MySQL 8.0 server under a new unified release model with the following features:
    - NDB 8.0 is developed in, built from, and released with the MySQL 8.0 source code tree.
    - The numbering scheme for NDB Cluster 8.0 releases follows the scheme for MySQL 8.0, starting with version 8.0.13.
    - Building the source with NDB support appends -cluster to the version string returned by mysql -V, as shown here:
      ```
      mysql -V
      mysql Ver 8.0.24-cluster for Linux on x86_64 (Source distribution)
      ```
    - NDB binaries continue to display both the MySQL Server version and the NDB engine version, like this:
      ```
      ndb_mgm -V
      MySQL distrib mysql-8.0.24 ndb-8.0.24, for Linux (x86_64)
      ```
    In MySQL Cluster NDB 8.0, these two version numbers are always the same.
    - To build the MySQL 8.0.13 (or later) source with NDB Cluster support, use the CMake option -DWITH_NDBCLUSTER.
  - **Platform support notes.** NDB 8.0 makes the following changes in platform support:
    - NDBCLUSTER no longer supports 32-bit platforms. Beginning with NDB 8.0.21, the NDB build process checks the system architecture and aborts if it is not a 64-bit platform.
    - Beginning with NDB 8.0.18, it is possible to build NDB from source for 64-bit ARM CPUs. Currently, this support is source-only, and we do not provide any precompiled binaries for this platform.
  - **Database and table names.** As of NDB 8.0.18, the 63-byte limit on identifiers for databases and tables is removed. These identifiers can now use up to 64 bytes, as for such objects using other MySQL storage engines. See Section 3.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.
  - **Generated names for foreign keys.** NDB (version 8.0.18 and later) now uses the pattern tbl_name_fk_N for naming internally generated foreign keys. This is similar to the pattern used by InnoDB.
  - **Schema and metadata distribution and synchronization.** NDB 8.0 makes use of the MySQL data dictionary to distribute schema information to SQL nodes joining a cluster and to synchronize new schema changes between existing SQL nodes. The following list describes individual enhancements relating to this integration work:
What is New in NDB Cluster 8.0

• **Schema distribution enhancements.** The NDB schema distribution coordinator, which handles schema operations and tracks their progress, has been extended in NDB 8.0.17 to ensure that resources used during a schema operation are released at its conclusion. Previously, some of this work was done by the schema distribution client; this has been changed due to the fact that the client did not always have all needed state information, which could lead to resource leaks when the client decided to abandon the schema operation prior to completion and without informing the coordinator.

To help fix this issue, schema operation timeout detection has been moved from the schema distribution client to the coordinator, providing the coordinator with an opportunity to clean up any resources used during the schema operation. The coordinator now checks ongoing schema operations for timeout at regular intervals, and marks participants that have not yet completed a given schema operation as failed when detecting timeout. It also provides suitable warnings whenever a schema operation timeout occurs. (It should be noted that, after such a timeout is detected, the schema operation itself continues.) Additional reporting is done by printing a list of active schema operations at regular intervals whenever one or more of these operations is ongoing.

As an additional part of this work, a new `mysqld` option `--ndb-schema-dist-timeout` makes it possible to set the length of time to wait until a schema operation is marked as having timed out.

• **Disk data file distribution.** Beginning with NDB Cluster 8.0.14, NDB uses the MySQL data dictionary to make sure that disk data files and related constructs such as tablespaces and log file groups are correctly distributed between all connected SQL nodes.

• **Schema synchronization of tablespace objects.** When a MySQL Server connects as an SQL node to an NDB cluster, it checks its data dictionary against the information found in the NDB dictionary.

Previously, the only NDB objects synchronized on connection of a new SQL node were databases and tables; MySQL NDB Cluster 8.0.14 and later also implement schema synchronization of disk data objects including tablespaces and log file groups. Among other benefits, this eliminates the possibility of a mismatch between the MySQL data dictionary and the NDB dictionary following a native backup and restore, in which tablespaces and log file groups were restored to the NDB dictionary, but not to the MySQL Server's data dictionary.

It is also no longer possible to issue a `CREATE TABLE` statement that refers to a nonexistent tablespace. Such a statement now fails with an error.

• **Database DDL synchronization enhancements.** Work done in NDB 8.0.17 insures that synchronization of databases by newly joined (or rejoined) SQL nodes with those on existing SQL nodes now makes proper use of the data dictionary so that any database-level operations (`CREATE DATABASE`, `ALTER DATABASE`, or `DROP DATABASE`) that may have been missed by this SQL node are now correctly duplicated on it when it connects (or reconnects) to the cluster.

As part of the schema synchronization procedure performed when starting, an SQL node now compares all databases on the cluster's data nodes with those in its own data dictionary, and if any of these is found to be missing from the SQL node's data dictionary, the SQL Node installs it locally by executing a `CREATE DATABASE` statement. A database thus created uses the default MySQL Server database properties (such as those as determined by `character_set_database` and `collation_database`) that are in effect on this SQL node at the time the statement is executed.

• **NDB metadata change detection and synchronization.** NDB 8.0.16 implements a new mechanism for detection of updates to metadata for data objects such as tables, tablespaces, and log file groups with the MySQL data dictionary. This is done using a thread, the NDB metadata
What is New in NDB Cluster 8.0

change monitor thread, which runs in the background and checks periodically for inconsistencies between the NDB dictionary and the MySQL data dictionary.

The monitor performs metadata checks every 60 seconds by default. The polling interval can be adjusted by setting the value of the ndb_metadata_check_interval system variable; polling can be disabled altogether by setting the ndb_metadata_check system variable to OFF. A status variable (also added in NDB 8.0.16) Ndb_metadata_detected_count shows the number of times since mysqld was last started that inconsistencies have been detected.

Beginning in version 8.0.18, NDB ensures that NDB table, log file group, and tablespace objects submitted by the metadata change monitor thread during operations following startup are automatically checked for mismatches and synchronized by the NDB binlog thread.

NDB 8.0.18 also adds two status variables relating to automatic synchronization:
Ndb_metadata_synced_count shows the number of objects synchronized automatically;
Ndb_metadata_excluded_count indicates the number of objects for which synchronization has failed (prior to NDB 8.0.22, this variable was named Ndb_metadata_blacklist_size). In addition, you can see which objects have been synchronized by inspecting the cluster log.

NDB 8.0.19 further enhances this functionality by adding databases to those objects in which changes are detected and synchronized. Only databases actually used by NDB tables are so handled; other databases which may be present in the MySQL data dictionary are ignored. This eliminates a previous requirement, for the case when a table existed in NDB but the table and the database to which it belonged did not exist on the SQL node, to create this database manually; now in such cases, the database and all NDB tables belonging to it should be created on the SQL node automatically.

NDB 8.0.19 also introduces the ndb_metadata_sync system variable; setting this variable to true overrides any settings that have been made for ndb_metadata_check_interval and ndb_metadata_check, causing the change monitor thread to begin continuous metadata change detection.

In NDB 8.0.22 and later, setting ndb_metadata_sync to true clears the list of objects for which synchronization has failed previously, which means it is no longer necessary to discover individual tables or to re-trigger synchronization by reconnecting the SQL node to the cluster. In addition, setting this variable to false clears the list of objects waiting to be retried.

Beginning with NDB 8.0.21, more detailed information about the current state of automatic synchronization than can be obtained from log messages or status variables is provided by two new tables added to the MySQL Performance Schema. The tables are listed here:

• ndb_sync_pending_objects: Contains information about database objects for which mismatches have been detected between the NDB dictionary and the MySQL data dictionary (and which have not been excluded from automatic synchronization).

• ndb_sync_excluded_objects: Contains information about NDB database objects which have been excluded because they cannot be synchronized between the NDB dictionary and the MySQL data dictionary, and thus require manual intervention.

A row in one of these tables provides the database object's parent schema, name, and type. Types of objects include schemas, tablespaces, log file groups, and tables. (If the object is a log file group or tablespace, the parent schema is NULL.) In addition, the ndb_sync_excluded_objects table shows the reason for which the object has been excluded.

These tables are present only if NDBCLUSTER storage engine support is enabled. For more information about these tables, see Performance Schema NDB Cluster Tables.

• Changes in NDB table extra metadata. In NDB 8.0.14 and later, the extra metadata property of an NDB table is used for storing serialized metadata from the MySQL data dictionary, rather than storing the binary representation of the table as in previous versions. (This was a .frm file, no
What is New in NDB Cluster 8.0

longer used by the MySQL Server—see MySQL Data Dictionary. As part of the work to support this change, the available size of the table’s extra metadata has been increased. This means that NDB tables created in NDB Cluster 8.0.14 and later are not compatible with previous NDB Cluster releases. Tables created in previous releases can be used with NDB 8.0.14 and later, but cannot be opened afterwards by an earlier version.

This metadata is accessible using the NDB API methods `getExtraMetadata()` and `setExtraMetadata()` that were implemented in NDB 8.0.13.

For more information, see Section 4.7, “Upgrading and Downgrading NDB Cluster”.

- On-the-fly upgrades of tables using .frm files. A table created in NDB 7.6 and earlier contains metadata in the form of a compressed .frm file, which is no longer supported in MySQL 8.0. To facilitate online upgrades to NDB 8.0, NDB performs on-the-fly translation of this metadata and writes it into the MySQL Server’s data dictionary, which enables the `mysqld` in NDB Cluster 8.0 to work with the table without preventing subsequent use of the table by a previous version of the NDB software.

  Important

Once a table’s structure has been modified in NDB 8.0, its metadata is stored using the data dictionary, and it can no longer be accessed by NDB 7.6 and earlier.

This enhancement also makes it possible to restore an NDB backup made using an earlier version to a cluster running NDB 8.0 (or later).

- Metadata consistency check error logging. As part of work previously done in NDB 8.0, the metadata check performed as part of auto-synchronization between the representation of an NDB table in the NDB dictionary and its counterpart in the MySQL data dictionary includes the table’s name, storage engine, and internal ID. Beginning with NDB 8.0.23, the range of properties checked is expanded to include properties of the following data objects:
  - Columns
  - Indexes
  - Foreign keys

In addition, details of any mismatches in metadata properties are now written to the MySQL server error log. The formats used for the error log messages differ slightly depending on whether the discrepancy is found on the table level or on the level of a column, index, or foreign key. The format for a log error resulting from a table-level property mismatch is shown here, where `property` is the property name, `ndb_value` is the property value as stored in the NDB dictionary, and `mysqld_value` is the value of the property as stored in the MySQL data dictionary:

```
Diff in 'property' detected, 'ndb_value' != 'mysqld_value'
```

For mismatches in properties of columns, indexes, and foreign keys, the format is as follows, where `obj_type` is one of column, index, or foreign key, and `obj_name` is the name of the object:

```
Diff in obj_type 'obj_name.property' detected, 'ndb_value' != 'mysqld_value'
```

Metadata checks are performed during automatic synchronization of NDB tables when they are installed in the data dictionary of any `mysqld` acting as an SQL node in an NDB Cluster. If the `mysqld` is debug-compiled, checks are also made whenever a `CREATE TABLE` statement is executed, and whenever an NDB table is opened.
What is New in NDB Cluster 8.0

- **Synchronization of user privileges with NDB_STORED_USER.** A new mechanism for sharing and synchronizing users, roles, and privileges between SQL nodes is available beginning with NDB 8.0.18, using the NDB_STORED_USER privilege. Distributed privileges as implemented in NDB 7.6 and earlier (see Distributed Privileges Using Shared Grant Tables) are no longer supported.

Once a user account is created on an SQL node, the user and its privileges can be stored in NDB and thus shared between all SQL nodes in the cluster by issuing a GRANT statement such as this one:

```sql
GRANT NDB_STORED_USER ON *.* TO 'jon'@'localhost';
```

NDB_STORED_USER always has global scope and must be granted using ON *.*. System reserved accounts such as mysql.session@localhost or mysql.infoschema@localhost cannot be assigned this privilege.

Roles can also be shared between SQL nodes by issuing the appropriate GRANT NDB_STORED_USER statement. Assigning such a role to a user does not cause the user to be shared; the NDB_STORED_USER privilege must be granted to each user explicitly.

A user or role having NDB_STORED_USER, along with its privileges, is shared with all SQL nodes as soon as they join a given NDB Cluster. Changes to the privileges of the user or role are synchronized immediately with all connected SQL nodes. It is possible to make such changes from any connected SQL node, but recommended practice is to do so from a designated SQL node only, since the order of execution of statements affecting privileges from different SQL nodes cannot be guaranteed to be the same on all SQL nodes.

**Implications for upgrades.** Due to changes in the MySQL server's privilege system (see Grant Tables), privilege tables using the NDB storage engine do not function correctly in NDB 8.0. It is safe but not necessary to retain such privilege tables created in NDB 7.6 or earlier, but they are no longer used for access control. Beginning with NDB 8.0.16, a mysqld acting as an SQL node and detecting such tables in NDB writes a warning to the MySQL server log, and creates InnoDB shadow tables local to itself; such shadow tables are created on each MySQL server connected to the cluster. When performing an upgrade from NDB 7.6 or earlier, the privilege tables using NDB can be removed safely using ndb_drop_table once all MySQL servers acting as SQL nodes have been upgraded (see Section 4.7, "Upgrading and Downgrading NDB Cluster").

The ndb_restore utility's --restore-privilege-tables option is deprecated but continues to be honored in NDB 8.0, and can still be used to restore distributed privilege tables present in a backup taken from a previous release of NDB Cluster to a cluster running NDB 8.0. These tables are handled as described in the preceding paragraph.

Shared users and grants are stored in the ndb_sql_metadata table, which in NDB 8.0.19 and later ndb_restore by default does not restore; you can specify the --include-stored-grants option to cause it to do so.

- **INFORMATION_SCHEMA changes.** The following changes are made in the display of information regarding Disk Data files in the INFORMATION_SCHEMA.FILES table:

  - Tablespaces and log file groups are no longer represented in the FILES table. (These constructs are not actually files.)
  
  - Each data file is now represented by a single row in the FILES table. Each undo log file is also now represented in this table by one row only. (Previously, a row was displayed for each copy of each of these files on each data node.)

In addition, INFORMATION_SCHEMA tables are now populated with tablespace statistics for MySQL Cluster tables. (Bug #27167728)

- **Error information with ndb_perror.** The deprecated --ndb option for perror has been removed. Instead, use ndb_perror to obtain error message information from NDB error codes. (Bug #81704, Bug #81705, Bug #23523926, Bug #23523957)
• **Condition pushdown enhancements.** Previously, condition pushdown was limited to predicate terms referring to column values from the same table to which the condition was being pushed. In NDB 8.0.16, this restriction is removed such that column values from tables earlier in the query plan can also be referred to from pushed conditions. As of NDB 8.0.18, joins comparing column expressions are supported, as are comparisons between columns in the same table. Columns and column expressions to be compared must be of exactly the same type; this means they must also be of the same signedness, length, character set, precision, and scale, whenever these attributes apply.

Pushing down larger parts of a condition allows more rows to be filtered out by the data nodes, thereby reducing the number of rows which *mysqld* must handle during join processing. Another benefit of these enhancements is that filtering can be performed in parallel in the LDM threads, rather than in a single *mysqld* process on an SQL node; this has the potential to improve query performance significantly.

Existing rules for type compatibility between column values being compared continue to apply (see Engine Condition Pushdown Optimization).

These additional improvements are made in NDB 8.0.21:

• Antijoins produced by the MySQL Optimizer through the transformation of `NOT EXISTS` and `NOT IN` queries (see Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations) can be pushed down to the data nodes by NDB.

This can be done when there is no unpushed condition on the table, and the query fulfills any other conditions which must be met for an outer join to be pushed down.

• NDB attempts to identify and evaluate a non-dependent scalar subquery before trying to retrieve any rows from the table to which it is attached. When it can do so, the value obtained is used as part of a pushed condition, instead of using the subquery which provided the value.

• **Increase in maximum row size.** NDB 8.0.18 increases the maximum number of bytes that can be stored in an NDBCLUSTER table from 14000 to 30000 bytes.

A BLOB or TEXT column continues to use 264 bytes of this total, as before.

The maximum offset for a fixed-width column of an NDB table is 8188 bytes; this is also unchanged from releases previous to 8.0.18.

See Section 3.7.5, “Limits Associated with Database Objects in NDB Cluster”, for more information.

• **ndb_mgm SHOW command and single user mode.** Beginning with NDB 8.0.17, when the cluster in single user mode, the output of the management client SHOW command indicates which API or SQL node has exclusive access while this mode is in effect.

• **Online column renames.** Beginning with NDB 8.0.18, columns of NDB tables can be renamed online, using ALGORITHM=INPLACE. See Section 7.11, “Online Operations with ALTER TABLE in NDB Cluster”, for more information.

• **Improved ndb_mgmd startup times.** Start times for management nodes daemon have been significantly improved in NDB 8.0.18 and later, in the following ways:

  • Due to replacing the list data structure formerly used by ndb_mgmd for handling node properties from configuration data with a hash table, overall startup times for the management server have been decreased by a factor of 6 or more.

  • In addition, in cases where data and SQL node host names not present in the management server’s hosts file are used in the cluster configuration file, ndb_mgmd start times can be up to 20 times shorter than was previously the case.
What is New in NDB Cluster 8.0

- **NDB API enhancements.** Beginning with NDB 8.0.18, NdbScanFilter::cmp() and several comparison methods of NdbInterpretedCode can be used to compare table column values with each other. The affected NdbInterpretedCode methods are listed here:
  - `branch_col_eq()`
  - `branch_col_ge()`
  - `branch_col_gt()`
  - `branch_col_le()`
  - `branch_col_lt()`
  - `branch_col_ne()`

  For all of the methods just listed, table column values to be compared must be of exactly matching types, including with respect to length, precision, signedness, scale, character set, and collation, as applicable.

  See the descriptions of the individual API methods for more information.

- **Offline multithreaded index builds.** It is now possible to specify a set of cores to be used for I/O threads performing offline multithreaded builds of ordered indexes, as opposed to normal I/O duties such as file I/O, compression, or decompression. “Offline” in this context refers to building of ordered indexes performed when the parent table is not being written to; such building takes place when an NDB cluster performs a node or system restart, or as part of restoring a cluster from backup using ndb_restore --rebuild-indexes.

  In addition, the default behaviour for offline index build work is modified to use all cores available to ndbmtd, rather limiting itself to the core reserved for the I/O thread. Doing so can improve restart and restore times and performance, availability, and the user experience.

  This enhancement is implemented as follows:

  1. The default value for BuildIndexThreads is changed from 0 to 128. This means that offline ordered index builds are now multithreaded by default.

  2. The default value for TwoPassInitialNodeRestartCopy is changed from false to true. This means that an initial node restart first copies all data from a “live” node to one that is starting—without creating any indexes—builds ordered indexes offline, and then again synchronizes its data with the live node, that is, synchronizing twice and building indexes offline between the two synchronizations. This causes an initial node restart to behave more like the normal restart of a node, and reduces the time required for building indexes.

  3. A new thread type (idxbld) is defined for the ThreadConfig configuration parameter, to allow locking of offline index build threads to specific CPUs.

  In addition, NDB now distinguishes the thread types that are accessible to ThreadConfig by these two criteria:

  1. Whether the thread is an execution thread. Threads of types main, ldm, recv, rep, tc, and send are execution threads; thread types io, watchdog, and idxbld are not.

  2. Whether the allocation of the thread to a given task is permanent or temporary. Currently all thread types except idxbld are permanent.

  For additional information, see the descriptions of the indicated parameters in the Manual. (Bug #25835748, Bug #26928111)
What is New in NDB Cluster 8.0

- **logbuffers table backup process information.** When performing an NDB backup, the `ndbinfo.logbuffers` table now displays information regarding buffer usage by the backup process on each data node. This is implemented as rows reflecting two new log types in addition to `REDO` and `DD-UNDO`. One of these rows has the log type `BACKUP-DATA`, which shows the amount of data buffer used during backup to copy fragments to backup files. The other row has the log type `BACKUP-LOG`, which displays the amount of log buffer used during the backup to record changes made after the backup has started. One each of these log type rows is shown in the `logbuffers` table for each data node in the cluster. Rows having these two log types are present in the table only while an NDB backup is currently in progress. (Bug #25822988)

- **ndbinfo.processes table on Windows.** The process ID of the monitor process used on Windows platforms by `RESTART` to spawn and restart a `mysqld` is now shown in the `processes` table as an `angel_pid`.

- **String hashing improvements.** Prior to NDB 8.0, all string hashing was based on first transforming the string into a normalized form, then MD5-hashing the resulting binary image. This could give rise to some performance problems, for the following reasons:
  
  - The normalized string is always space padded to its full length. For a `VARCHAR`, this often involved adding more spaces than there were characters in the original string.
  
  - The string libraries were not optimized for this space padding, which added considerable overhead in some use cases.
  
  - The padding semantics varied between character sets, some of which were not padded to their full length.
  
  - The transformed string could become quite large, even without space padding; some Unicode 9.0 collations can transform a single code point into 100 bytes or more of character data.
  
  - Subsequent MD5 hashing consisted mainly of padding with spaces, and was not particularly efficient, possibly causing additional performance penalties by flushing significant portions of the L1 cache.

  A collation provides its own hash function, which hashes the string directly without first creating a normalized string. In addition, for a Unicode 9.0 collation, the hash is computed without padding. NDB now takes advantage of this built-in function whenever hashing a string identified as using a Unicode 9.0 collation.

  Since, for other collations, there are existing databases which are hash partitioned on the transformed string, NDB continues to employ the previous method for hashing strings that use these, to maintain compatibility. (Bug #89590, Bug #89604, Bug #89609, Bug #27515000, Bug #27523758, Bug #27522732)

- **RESET MASTER changes.** Because the MySQL Server now executes `RESET MASTER` with a global read lock, the behavior of this statement when used with NDB Cluster has changed in the following two respects:

  - It is no longer guaranteed to be synchronous; that is, it is now possible that a read coming immediately before `RESET MASTER` is issued may not be logged until after the binary log has been rotated.
  
  - It now behaves in exactly the same fashion, whether the statement is issued on the same SQL node that is writing the binary log, or on a different SQL node in the same cluster.

  **Note**

  `SHOW BINLOG EVENTS`, `FLUSH LOGS`, and most data definition statements continue, as they did in previous NDB versions, to operate in a synchronous fashion.
What is New in NDB Cluster 8.0

- **ndb_restore option usage.** Beginning with NDB 8.0.16, the `--nodeid` and `--backupid` options are both required when invoking `ndb_restore`.

- **ndb_log_bin default.** Beginning with NDB 8.0.16, the default value of the `ndb_log_bin` system variable has changed from `TRUE` to `FALSE`.

- **Dynamic transactional resource allocation.** Allocation of resources in the transaction coordinator (see The DBTC Block) is now performed using dynamic memory pools. This means that resource allocation determined by data node configuration parameters such as `MaxDMLOperationsPerTransaction`, `MaxNoOfConcurrentIndexOperations`, `MaxNoOfConcurrentOperations`, `MaxNoOfConcurrentScans`, `MaxNoOfConcurrentTransactions`, `MaxNoOfFiredTriggers`, `MaxNoOfLocalScans`, and `TransactionBufferMemory` is now done in such a way that, if the load represented by each of these parameters is within the target load for all such resources, others of these resources can be limited so as not to exceed the total resources available.

As part of this work, several new data node parameters controlling transactional resources in DBTC, listed here, have been added:

- `ReservedConcurrentIndexOperations`
- `ReservedConcurrentOperations`
- `ReservedConcurrentScans`
- `ReservedConcurrentTransactions`
- `ReservedFiredTriggers`
- `ReservedLocalScans`
- `ReservedTransactionBufferMemory`

See the descriptions of the parameters just listed for further information.

- **Backups using multiple LDMs per data node.** NDB backups can now be performed in a parallel fashion on individual data nodes using multiple local data managers (LDMs). (Previously, backups were done in parallel across data nodes, but were always serial within data node processes.) No special syntax is required for the `START BACKUP` command in the `ndb_mgm` client to enable this feature, but all data nodes must be using multiple LDMs. This means that data nodes must be running `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; you can do this by choosing an appropriate setting for one of the multi-threaded data node configuration parameters `MaxNoOfExecutionThreads` or `ThreadConfig`.

Backups using multiple LDMs create subdirectories, one per LDM, under the `BACKUP/BACKUP-<backup_id>/` directory. `ndb_restore` now detects these subdirectories automatically, and if they exist, attempts to restore the backup in parallel; see Section 6.23.3, “Restoring from a backup taken in parallel”, for details. (Single-threaded backups are restored as in previous versions of NDB.) It is also possible to restore backups taken in parallel using an `ndb_restore` binary from a previous version of NDB Cluster by modifying the usual restore procedure; Section 6.23.3.2, “Restoring a parallel backup serially”, provides information on how to do this.

- **Binary configuration file enhancements.** Beginning with NDB 8.0.18, a new format is used for the management server's binary configuration file. Previously, a maximum of 16381 sections could appear in the cluster configuration file; now the maximum number of sections is 4G. This is intended to support larger numbers of nodes in a cluster than was possible before this change.

Upgrades to the new format are relatively seamless, and should seldom if ever require manual intervention, as the management server continues to be able to read the old format without issue. A downgrade from NDB 8.0.18 (or later) to an older version of the NDB Cluster software requires
manual removal of any binary configuration files or, alternatively, starting the older management server binary with the `--initial` option.

For more information, see Section 4.7, “Upgrading and Downgrading NDB Cluster”.

- **Increased number of data nodes.** NDB 8.0.18 increases the maximum number of data nodes supported per cluster to 144 (previously, this was 48). Data nodes can now use node IDs in the range 1 to 144, inclusive.

  Previously, the recommended node IDs for management nodes were 49 and 50. These are still supported for management nodes, but using them as such limits the maximum number of data nodes to 142; for this reason, it is now recommended that node IDs 145 and 146 are used for management nodes.

  As part of this work, the format used for the data node `sysfile` has been updated to version 2. This file records information such as the last global checkpoint index, restart status, and node group membership of each node (see NDB Cluster Data Node File System Directory).

- **RedoOverCommitCounter and RedoOverCommitLimit changes.** Due to ambiguities in the semantics for setting them to 0, the minimum value for each of the data node configuration parameters `RedoOverCommitCounter` and `RedoOverCommitLimit` has been increased to 1, beginning with NDB 8.0.19.

- **ndb_autoincrement_prefetch_sz changes.** In NDB 8.0.19, the default value of the `ndb_autoincrement_prefetch_sz` server system variable is increased to 512.

- **Changes in parameter maximums and defaults.** NDB 8.0.19 makes the following changes in configuration parameter maximum and default values:
  - The maximum for `DataMemory` is increased to 16 terabytes.
  - The maximum for `DiskPageBufferMemory` is also increased to 16 terabytes.
  - The default value for `StringMemory` is increased to 25%.
  - The default for `LcpScanProgressTimeout` is increased to 180 seconds.

- **Disk Data checkpointing improvements.** NDB Cluster 8.0.19 provides a number of new enhancements which help to reduce the latency of checkpoints of Disk Data tables and tablespaces when using non-volatile memory devices such as solid-state drives and the NVMe specification for such devices. These improvements include those in the following list:
  - Avoiding bursts of checkpoint disk writes
  - Speeding up checkpoints for disk data tablespaces when the redo log or the undo log becomes full
  - Balancing checkpoints to disk and in-memory checkpoints against one other, when necessary
  - Protecting disk devices from overload to help ensure low latency under high loads

  As part of this work, NDB 8.0.19 introduces two new data node configuration parameters. `MaxDiskDataLatency` places a ceiling on the degree of latency permitted for disk access and causes transactions taking longer than this length of time to be aborted. `DiskDataUsingSameDisk` makes it possible to take advantage of housing Disk Data tablespaces on separate disks by increasing the rate at which checkpoints of such tablespaces can be performed.

  In addition, three new tables in the `ndbinfo` database, also added in NDB 8.0.19 and listed here, provide information about Disk Data performance:
  - The `diskstat` table reports on writes to Disk Data tablespaces during the past second
What is New in NDB Cluster 8.0

- The `diskstats_1sec` table reports on writes to Disk Data tablespaces for each of the last 20 seconds.
- The `pgman_time_track_stats` table reports on the latency of disk operations relating to Disk Data tablespaces.

**Memory allocation and TransactionMemory.** NDB 8.0.19 introduces a new `TransactionMemory` parameter which simplifies allocation of data node memory for transactions as part of the work done to pool transactional and Local Data Manager (LDM) memory. This parameter is intended to replace several older transactional memory parameters which have been deprecated.

Transaction memory can now be set in any of the three ways listed here:

- Several configuration parameters are incompatible with `TransactionMemory`. If any of these are set, `TransactionMemory` cannot be set (see Parameters incompatible with TransactionMemory), and the data node’s transaction memory is determined as it was previous to NDB 8.0.19.

  **Note**
  Attempting to set `TransactionMemory` and any of these parameters concurrently in the `config.ini` file prevents the management server from starting.

- If `TransactionMemory` is set, this value is used for determining transaction memory. `TransactionMemory` cannot be set if any of the incompatible parameters mentioned in the previous item have also been set.

- If none of the incompatible parameters are set and `TransactionMemory` is also not set, transaction memory is set by NDB.

For more information, see the description of `TransactionMemory`, as well as Section 5.3.13, "Data Node Memory Management".

- **Support for additional fragment replicas.** NDB 8.0.19 increases the maximum number of fragment replicas supported in production from two to four. (Previously, it was possible to set `NoOfReplicas` to 3 or 4, but this was not officially supported or verified in testing.)

- **Restoring by slices.** Beginning with NDB 8.0.20, it is possible to divide a backup into roughly equal portions (slices) and to restore these slices in parallel using two new options implemented for `ndb_restore`:
  - `--num-slices` determines the number of slices into which the backup should be divided.
  - `--slice-id` provides the ID of the slice to be restored by the current instance of `ndb_restore`.

  This makes it possible to employ multiple instances of `ndb_restore` to restore subsets of the backup in parallel, potentially reducing the amount of time required to perform the restore operation.

  For more information, see the description of the `ndb_restore --num-slices` option.

- **Read from any fragment replica enabled.** Beginning with NDB 8.0.19, read from any fragment replica is enabled by default for all NDB tables. This means that the default value for the `ndb_read_backup` system variable is now ON, and that the value of the `NDB_TABLE` comment option `READ_BACKUP` is 1 when creating a new NDB table. Enabling read from any fragment replica significantly improves performance for reads from NDB tables, with minimal impact on writes.

  For more information, see the description of the `ndb_read_backup` system variable, and Setting NDB_TABLE Options.
What is New in NDB Cluster 8.0

- **ndb_blob_tool enhancements.** Beginning with NDB 8.0.20, the ndb_blob_tool utility can detect missing blob parts for which inline parts exist and replace these with placeholder blob parts (consisting of space characters) of the correct length. To check whether there are missing blob parts, use the `--check-missing` option with this program. To replace any missing blob parts with placeholders, use the `--add-missing` option.

  For more information, see Section 6.6, "ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables".

- **ndbinfo versioning.** NDB 8.0.20 and later supports versioning for ndbinfo tables, and maintains the current definitions for its tables internally. At startup, NDB compares its supported ndbinfo version with the version stored in the data dictionary. If the versions differ, NDB drops any old ndbinfo tables and recreates them using the current definitions.

- **Support for Fedora Linux.** Beginning with NDB 8.0.20, Fedora Linux is a supported platform for NDB Cluster Community releases and can be installed using the RPMs supplied for this purpose by Oracle. These can be obtained from the NDB Cluster downloads page.

- **NDB programs—NDBT dependency removal.** The dependency of a number of NDB utility programs on the NDBT library has been removed. This library is used internally for development, and is not required for normal use; its inclusion in these programs could lead to unwanted issues when testing.

  Affected programs are listed here, along with the NDB versions in which the dependency was removed:

  - `ndb_restore`, in NDB 8.0.17
  - `ndb_delete_all`, in NDB 8.0.18
  - `ndb_show_tables`, in NDB 8.0.20
  - `ndb_waiter`, in NDB 8.0.20

  The principal effect of this change for users is that these programs no longer print `NDBT_ProgramExit - status` following completion of a run. Applications that depend upon such behavior should be updated to reflect the change when upgrading to the indicated versions.

- **Pushdown of outer joins and semijoins.** Work done in NDB 8.0.20 allows many outer joins and semijoins, and not only those using a primary key or unique key lookup, to be pushed down to the data nodes (see Engine Condition Pushdown Optimization).

  Outer joins using scans which can now be pushed include those which meet the following conditions:

  - There are no unpushed conditions on the table
  - There are no unpushed conditions on other tables in the same join nest, or in upper join nests on which it depends
  - All other tables in the same join nest, or in upper join nests on which it depends, are also pushed

  A semijoin that uses an index scan can now be pushed if it meets the the conditions just noted for a pushed outer join, and it uses the `firstMatch` strategy (see Optimizing IN and EXISTS Subquery Predicates with Semijoin Transformations).

  When a join cannot be pushed, `EXPLAIN` should provide the reason or reasons.

- **Foreign keys and lettercasing.** NDB stores the names of foreign keys using the case with which they were defined. Formerly, when the value of the `lower_case_table_names` system variable was set to 0, it performed case-sensitive comparisons of foreign key names as used in `SELECT` and other SQL statements with the names as stored. Beginning with NDB 8.0.20, such
comparisons are now always performed in a case-insensitive fashion, regardless of the value of `lower_case_table_names`.

- **Multiple transporters.** NDB 8.0.20 introduces support for multiple transporters to handle node-to-node communication between pairs of data nodes. This facilitates higher rates of update operations for each node group in the cluster, and helps avoid constraints imposed by system or other limitations on inter-node communications using a single socket.

  By default, NDB now uses a number of transporters based on the number of local data management (LDM) threads or the number of transaction coordinator (TC) threads, whichever is greater. By default, the number of transporters is equal to half of this number. While the default should perform well for most workloads, it is possible to adjust the number of transporters employed by each node group by setting the `NodeGroupTransporters` data node configuration parameter (also introduced in NDB 8.0.20), up a maximum of the greater of the number of LDM threads or the number of TC threads. Setting it to 0 causes the number of transporters to be the same as the number of LDM threads.

- **ndb_restore: primary key schema changes.** NDB 8.0.21 (and later) supports different primary key definitions for source and target tables when restoring an NDB native backup with `ndb_restore` when it is run with the `--allow-pk-changes` option. Both increasing and decreasing the number of columns making up the original primary key are supported.

  When the primary key is extended with an additional column or columns, any columns added must be defined as `NOT NULL`, and no values in any such columns may be changed during the time that the backup is being taken. Because some applications set all column values in a row when updating it, whether or not all values are actually changed, this can cause a restore operation to fail even if no values in the column to be added to the primary key have changed. You can override this behavior using the `--ignore-extended-pk-updates` option also added in NDB 8.0.21; in this case, you must ensure that no such values are changed.

  A column can be removed from the table's primary key whether or not this column remains part of the table.

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

- **Merging backups with ndb_restore.** In some cases, it may be desirable to consolidate data originally stored in different instances of NDB Cluster (all using the same schema) into a single target NDB Cluster. This is now supported when using backups created in the `ndb_mgm` client (see Section 7.8.2, “Using The NDB Cluster Management Client to Create a Backup”) and restoring them with `ndb_restore`, using the `--remap-column` option added in NDB 8.0.21 along with `--restore-data` (and possibly additional compatible options as needed or desired). `--remap-column` can be employed to handle cases in which primary and unique key values are overlapping between source clusters, and it is necessary that they do not overlap in the target cluster, as well as to preserve other relationships between tables such as foreign keys.

  `--remap-column` takes as its argument a string having the format `db.tbl.col:fn:args`, where `db`, `tbl`, and `col` are, respectively, the names of the database, table, and column, `fn` is the name of a remapping function, and `args` is one or more arguments to `fn`. There is no default value. Only `offset` is supported as the function name, with `args` as the integer offset to be applied to the value of the column when inserting it into the target table from the backup. This column must be one of `INT`
What is New in NDB Cluster 8.0

or BIGINT; the allowed range of the offset value is the same as the signed version of that type (this allows the offset to be negative if desired).

The new option can be used multiple times in the same invocation of ndb_restore, so that you can remap to new values multiple columns of the same table, different tables, or both. The offset value does not have to be the same for all instances of the option.

In addition, two new options are provided for ndb_desc, also beginning in NDB 8.0.21:

- --auto-inc (short form -a): Includes the the next auto-increment value in the output, if the table has an AUTO_INCREMENT column.
- --context (short form -x): Provides extra information about the table, including the schema, database name, table name, and internal ID.

For more information and examples, see the description of the --remap-column option.

- **Send thread improvements.** As of NDB 8.0.20, each send thread now handles sends to a subset of transporters, and each block thread now assists only one send thread, resulting in more send threads, and thus better performance and data node scalability.

- **Adaptive spin control using SpinMethod.** A simple interface for setting up adaptive CPU spin on platforms supporting it, using the SpinMethod data node parameter. This parameter (added in NDB 8.0.20, functional beginning with NDB 8.0.24) has four settings, one each for static spinning, cost-based adaptive spinning, latency-optimized adaptive spinning, and adaptive spinning optimized for database machines on which each thread has its own CPU. Each of these settings causes the data node to use a set of predetermined values for one or more spin parameters which enable adaptive spinning, set spin timing, and set spin overhead, as appropriate to a given scenario, thus obviating the need to set these directly for common use cases.

  For fine-tuning spin behavior, it is also possible to set these and additional spin parameters directly, using the existing SchedulerSpinTimer data node configuration parameter as well as the following DUMP commands in the ndb_mgm client:

  - DUMP 104000 (SetSchedulerSpinTimerAll): Sets spin time for all threads
  - DUMP 104001 (SetSchedulerSpinTimerThread): Sets spin time for a specified thread
  - DUMP 104002 (SetAllowedSpinOverhead): Sets spin overhead as the number of units of CPU time allowed to gain 1 unit of latency
  - DUMP 104003 (SetSpintimePerCall): Sets the time for a call to spin
  - DUMP 104004 (EnableAdaptiveSpinning): Enables or disables adaptive spinning

  NDB 8.0.20 also adds a new TCP configuration parameter TcpSpinTime which sets the time to spin for a given TCP connection.

  The ndb_top tool is also enhanced to provide spin time information per thread.

  For additional information, see the description of the SpinMethod parameter, the listed DUMP commands, and Section 6.29, “ndb_top — View CPU usage information for NDB threads”.

- **Disk Data and cluster restarts.** Beginning with NDB 8.0.21, an initial restart of the cluster forces the removal of all Disk Data objects such as tablespaces and log file groups, including any data files and undo log files associated with these objects.

  See Section 7.10, “NDB Cluster Disk Data Tables”, for more information.
• **Disk Data extent allocation.** Beginning with NDB 8.0.20, allocation of extents in data files is done in a round-robin fashion among all data files used by a given tablespace. This is expected to improve distribution of data in cases where multiple storage devices are used for Disk Data storage.

For more information, see Section 7.10.1, “NDB Cluster Disk Data Objects”.

• **--ndb-log-fail-terminate option.** Beginning with NDB 8.0.21, you can cause the SQL node to terminate whenever it is unable to log all row events fully. This can be done by starting `mysqld` with the `--ndb-log-fail-terminate` option.

• **AllowUnresolvedHostNames parameter.** By default, a management node refuses to start when it cannot resolve a host name present in the global configuration file, which can be problematic in some environments such as Kubernetes. Beginning with NDB 8.0.22, it is possible to override this behavior by setting `AllowUnresolvedHostNames` to `true` in the `[tcp default]` section of the cluster global configuration file (config.ini file). Doing so causes such errors to be treated as warnings instead, and to permit `ndb_mgmd` to continue starting.

• **Blob write performance enhancements.** NDB 8.0.22 implements a number of improvements which allow more efficient batching when modifying multiple blob columns in the same row, or when modifying multiple rows containing blob columns in the same statement, by reducing the number of round trips required between an SQL or other API node and the data nodes when applying these modifications. The performance of many `INSERT`, `UPDATE`, and `DELETE` statements can thus be improved. Examples of such statements are listed here, where `table` is an NDB table containing one or more Blob columns:

- **INSERT INTO table VALUES ROW(1, blob_value1, blob_value2, ...), that is, insertion of a row containing one or more Blob columns**

- **INSERT INTO table VALUES ROW(1, blob_value1), ROW(2, blob_value2), ROW(3, blob_value3), ..., that is, insertion of multiple rows containing one or more Blob columns**

- **UPDATE table SET blob_column1 = blob_value1, blob_column2 = blob_value2, ...**

- **UPDATE table SET blob_column = blob_value WHERE primary_key_column in (value_list), where the primary key column is not a Blob type**

- **DELETE FROM table WHERE primary_key_column = value, where the primary key column is not a Blob type**

- **DELETE FROM table WHERE primary_key_column IN (value_list), where the primary key column is not a Blob type**

Other SQL statements may benefit from these improvements as well. These include `LOAD DATA INFILE` and `CREATE TABLE ... SELECT ...`. In addition, `ALTER TABLE table ENGINE = NDB`, where `table` uses a storage engine other than NDB prior to execution of the statement, may also execute more efficiently.

This enhancement applies to statements affecting columns of MySQL type `BLOB`, `MEDIUMBLOB`, `LONGBLOB`, `TEXT`, `MEDIUMTEXT`, and `LONGTEXT`. Statements which update `TINYBLOB` or `TINYTEXT` columns (or both types) only are not affected by this work, and no changes in their performance should be expected.

The performance of some SQL statements is not noticeably improved by this enhancement, due to the fact that they require scans of table Blob columns, which breaks up batching. Such statements include those of the types listed here:

- **SELECT FROM table [WHERE key_column IN (blob_value_list)], where rows are selected by matching on a primary key or unique key column which uses a Blob type**
• **UPDATE** table **SET** blob_column = blob_value **WHERE** condition, using a condition which does not depend on a unique value

• **DELETE FROM** table **WHERE** condition to delete rows containing one or more Blob columns, using a condition which does not depend on a unique value

• A copying **ALTER TABLE** statement on a table which already used the NDB storage engine prior to executing the statement, and whose rows contain one or more Blob columns before or after the statement is executed (or both)

To take advantage of this improvement to its fullest extent, you may wish to increase the values used for the **--ndb-batch-size** and **--ndb-blob-write-batch-bytes** options for mysqld, to minimize the number of round trips required to modify Blobs. For replication, it is also recommended that you enable the **slave_allow_batching** system variable, which minimizes the number of round trips required by the replica cluster to apply epoch transactions.

• **Node.js update.** Beginning with with NDB 8.0.22, the NDB adapter for Node.js is built using version 12.18.3, and only that version (or a later version of Node.js) is now supported.

• **Encrypted backups.** NDB 8.0.22 adds support for backup files encrypted using AES-256-CBC; this is intended to protect against recovery of data from backups that have been accessed by unauthorized parties. When encrypted, backup data is protected by a user-supplied password. The password can be any string consisting of up to 256 characters from the range of printable ASCII characters other than !, ', " , $, %, \, and ^. Retention of the password used to encrypt any given NDB Cluster backup must be performed by the user or application; NDB does not save the password. The password can be empty, although this is not recommended.

When taking an NDB Cluster backup, you can encrypt it by using `ENCRYPT PASSWORD=password` with the management client `START BACKUP` command. Users of the MGM API can also initiate an encrypted backup by calling `ndb_mgm_start_backup4()`.

You can encrypt existing backup files using the `ndbxfrm` utility which is added to the NDB Cluster distribution in the 8.0.22 release; this program can also be employed for decrypting encrypted backup files. In addition, `ndbxfrm` can compress backup files and decompress compressed backup files using the same method that is employed by NDB Cluster for creating backups when the `CompressedBackup` configuration parameter is set to 1.

To restore from an encrypted backup, use `ndb_restore` with the options **--decrypt** and **--backup-password**. Both options are required, along with any others that would be needed to restore the same backup if it were not encrypted. `ndb_print_backup_file` and `ndbxfrm` can also read encrypted files using, respectively, `-P password` and **--decrypt-password=password**.

In all cases in which a password is supplied together with an option for encryption or decryption, the password must be quoted; you can use either single or double quotation marks to delimit the password.

Beginning with NDB 8.0.24, several NDB programs, listed here, also support input of the password from standard input, similarly to how this is done when logging in interactively with the `mysql` client using the **--password** option (without including the password on the command line):

• For `ndb_restore` and `ndb_print_backup_file`, the **--backup-password-from-stdin** option enables input of the password in a secure fashion, similar to how it is done by the `mysql` client `--password` option. For `ndb_restore`, use the option together with the **--decrypt** option; for `ndb_print_backup_file`, use the option in place of the `-P` option.

• For `ndb_mgm` the option **--backup-password-from-stdin**, is supported together with **--execute "START BACKUP [options]"** for starting a cluster backup from the system shell.
• **Two ndbxfrm options, --encrypt-password-from-stdin and --decrypt-password-from-stdin**, cause similar behavior when using that program to encrypt or to decrypt a backup file.

See the descriptions of the programs just listed for more information.

It is also possible, beginning with NDB 8.0.22, to enforce encryption of backups by setting `RequireEncryptedBackup=1` in the `[ndbd default]` section of the cluster global configuration file. When this is done, the `ndb_mgm` client rejects any attempt to perform a backup that is not encrypted.

Beginning with NDB 8.0.24, you can cause `ndb_mgm` to use encryption whenever it creates a backup by starting it with `--encrypt-backup`. In this case, the user is prompted for a password when invoking `START BACKUP` if none is supplied.

• **IPv6 support.** Beginning with NDB 8.0.22, IPv6 addressing is supported for connections to management and data nodes; this includes connections between management and data nodes with SQL nodes. When configuring a cluster, you can use numeric IPv6 addresses, host names which resolve to IPv6 addresses or both.

For IPv6 addressing to work, the operating platform and network on which the cluster is deployed must support IPv6. As when using IPv4 addressing, hostname resolution to IPv6 addresses must be provided by the operating platform.

IPv4 addressing continues to be supported by NDB. Using IPv4 and IPv6 addresses concurrently is not recommended, but can be made to work in the following cases:

- When the management node is configured with IPv6 and data nodes are configured with IPv4 addresses in the `config.ini` file: This works if `--bind-address` is not used with mgmd, and data nodes are started with `--ndb-connectstring` set to the IPv4 address of the management nodes.

- When the management node is configured with IPv4 and data nodes are configured with IPv6 addresses in `config.ini`: Similarly to the other case, this works if `--bind-address` is not passed to mgmd and data nodes are started with `--ndb-connectstring` set to the IPv6 address of the management node.

These cases work because `ndb_mgmd` does not bind to any IP address by default.

To perform an upgrade from a version of NDB that does not support IPv6 addressing to one that does, provided that the network supports IPv4 and IPv6, first perform the software upgrade; after this has been done, you can update IPv4 addresses used in the `config.ini` file with IPv6 addresses. After this, to cause the configuration changes to take effect and to make the cluster start using the IPv6 addresses, it is necessary to perform a system restart of the cluster.

• **Auto-Installer deprecation and removal.** The MySQL NDB Cluster Auto-Installer web-based installation tool (`ndb_setup.py`) is deprecated in NDB 8.0.22, and is removed in NDB 8.0.23 and later. It is no longer supported.

• **ndbmemcache deprecation and removal.** `ndbmemcache` is no longer supported. `ndbmemcache` was deprecated in NDB 8.0.22, and removed in NDB 8.0.23.

• **ndbinfo backup_id table.** NDB 8.0.24 adds a `backup_id` table to the `ndbinfo` information database. This is intended to serve as a replacement for obtaining this information by using `ndb_select_all` to dump the contents of the internal `SYSTAB_0` table, which is error-prone and takes an excessively long time to perform.

This table has a single column and row containing the ID of the most recent backup of the cluster taken using the `START BACKUP` management client command. In the event that no backup of this cluster can be found, the table contains a single row whose column value is `0`.
What is New in NDB Cluster 8.0

• **Table partitioning enhancements.** NDB 8.0.23 introduces a new method for handling table partitions and fragments, which can determine the number of local data managers (LDMs) for a given data node independently of the number of redo log parts. This means that the number of LDMs can now be highly variable. NDB can employ this method when the `ClassicFragmentation` data node configuration parameter, also implemented in NDB 8.0.23, is set to `false`; when this is the case, the number of LDMs is no longer used to determine how many partitions to create for a table per data node, and the value of the `PartitionsPerNode` parameter (also introduced in NDB 8.0.23) determines this number instead, which is also used for calculating the number of fragments used for a table.

When `ClassicFragmentation` has its default value `true`, then the traditional method of using the number of LDMs is used to determine the number of fragments that a table should have.

For more information, see the descriptions of the new parameters referenced previously, in Multi-Thread Configuration Parameters (ndbmtd).

• **Terminology updates.** To align with work begun in MySQL 8.0.21 and NDB 8.0.21, NDB 8.0.23 implements a number of changes in terminology, listed here:

  • The system variable `ndb_slave_conflict_role` is now deprecated. It is replaced by `ndb_conflict_role`.

  • Many NDB status variables are deprecated. These variables, and their replacements, are shown in the following table:

    | Deprecated variable                                         | Replacement                                          |
    |-------------------------------------------------------------|------------------------------------------------------|
    | `Ndb_api_adaptive_send_deferred_count_slave`               | `Ndb_api_adaptive_send_deferred_count_replica`       |
    | `Ndb_api_adaptive_send_forced_count_slave`                 | `Ndb_api_adaptive_send_forced_count_replica`         |
    | `Ndb_api_adaptive_send_unforced_count_slave`               | `Ndb_api_adaptive_send_unforced_count_replica`       |
    | `Ndb_api_bytes_received_count_slave`                       | `Ndb_api_bytes_received_count_replica`               |
    | `Ndb_api_bytes_sent_count_slave`                           | `Ndb_api_bytes_sent_count_replica`                   |
    | `Ndb_api_pk_op_count_slave`                                | `Ndb_api_pk_op_count_replica`                        |
    | `Ndb_api_pruned_scan_count_slave`                          | `Ndb_api_pruned_scan_count_replica`                  |
    | `Ndb_api_range_scan_count_slave`                           | `Ndb_api_range_scan_count_replica`                   |
    | `Ndb_api_read_row_count_slave`                             | `Ndb_api_read_row_count_replica`                     |
    | `Ndb_api_scan_batch_count_slave`                           | `Ndb_api_scan_batch_count_replica`                   |
    | `Ndb_api_table_scan_count_slave`                           | `Ndb_api_table_scan_count_replica`                   |
    | `Ndb_api_trans_abort_count_slave`                          | `Ndb_api_trans_abort_count_replica`                  |
    | `Ndb_api_trans_close_count_slave`                          | `Ndb_api_trans_close_count_replica`                  |
    | `Ndb_api_trans_commit_count_slave`                         | `Ndb_api_trans_commit_count_replica`                 |
    | `Ndb_api_trans_local_read_row_count_slave`                 | `Ndb_api_trans_local_read_row_count_replica`         |
    | `Ndb_api_trans_start_count_slave`                          | `Ndb_api_trans_start_count_replica`                  |
    | `Ndb_api_uk_op_count_slave`                                | `Ndb_api_uk_op_count_replica`                        |
    | `Ndb_api_wait_exec_complete_count_slave`                   | `Ndb_api_wait_exec_complete_count_replica`           |
    | `Ndb_api_wait_meta_request_count_slave`                    | `Ndb_api_wait_meta_request_count_replica`           |
    | `Ndb_api_wait_nanos_count_slave`                           | `Ndb_api_wait_nanos_count_replica`                   |
    | `Ndb_api_wait_scan_result_count_slave`                     | `Ndb_api_wait_scan_result_count_replica`            |
What is New in NDB Cluster 8.0

<table>
<thead>
<tr>
<th>Deprecated variable</th>
<th>Replacement</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ndb_slave_max_replicated_epoch</td>
<td>Ndb_replica_max_replicated_epoch</td>
</tr>
</tbody>
</table>

The deprecated status variables continue to be shown in the output of `SHOW STATUS`, but applications should be updated as soon as possible not to rely upon them any longer, since their availability in future release series is not guaranteed.

- The values `ADD_TABLE_MASTER` and `ADD_TABLE_SLAVE` previously shown in the `tab_copy_status` column of the `ndbinfo.ndbinfo.table_distribution_status` table are deprecated. These are replaced by, respectively, the values `ADD_TABLE_COORDINATOR` and `ADD_TABLE_PARTICIPANT`.

- The `--help` output of some NDB client and utility programs such as `ndb_restore` has been modified.

- **ThreadConfig enhancements.** As of NDB 8.0.23, the configurability of the `ThreadConfig` parameter has been extended with two new thread types, listed here:
  - `query`: A query thread works (only) on `READ COMMITTED` queries. A query thread also acts as a recovery thread. The number of query threads must be 0, 1, 2, or 3 times the number of LDM threads. 0 (the default, unless using `ThreadConfig`, or `AutomaticThreadConfig` is enabled) causes LDMs to behave as they did prior to NDB 8.0.23.
  - `recover`: A recovery thread retrieves data from a local checkpoint. A recovery thread specified as such never acts as a query thread.

It is also possible to combine the existing `main` and `rep` threads in either of two ways:

- Into a single thread by setting either one of these arguments to 0. When this is done, the resulting combined thread is shown with the name `main_rep` in the `ndbinfo.threads` table.
- Together with the `recv` thread by setting both `ldm` and `tc` to 0, and setting `recv` to 1. In this case, the combined thread is named `main_rep_recv`.

In addition, the maximum numbers of a number of existing thread types have been increased. The new maximums, including those for query threads and recovery threads, are listed here:

- LDM: 332
- Query: 332
- Recovery: 332
- TC: 128
- Receive: 64
- Send: 64
- Main: 2

Maximums for other thread types remain unchanged.

For more information, see the descriptions of the `ThreadConfig` parameter and the `ndbinfo.threads` table.

Also, as the result of work done relating to this task, NDB now employs mutexes to protect job buffers when using more than 32 block threads. While this can cause a slight decrease in performance (1 to 2 percent in most cases), it also significantly reduces the amount of memory required by very large configurations. For example, a setup with 64 threads which used 2 GB of job buffer memory prior
to NDB 8.0.23 should require only about 1 GB instead in NDB 8.0.23 and later. In our testing this has resulted in an overall improvement on the order of 5 percent in the execution of very complex queries.

- **ndbmtd Thread Auto-Configuration.** Beginning with NDB 8.0.23, it is possible to employ automatic configuration of threads for multi-threaded data nodes using the `ndbmtd` configuration parameter `AutomaticThreadConfig`. When this parameter is set to 1, NDB sets up thread assignments automatically, based on the number of processors available to applications, for all thread supported thread types, including the new `query` and `recover` thread types described in the previous item. If the system does not limit the number of processors, you can do so if desired by setting `NumCPUs` (also added in NDB 8.0.23). Otherwise, automatic thread configuration accommodates up to 1024 CPUs.

  Automatic thread configuration occurs regardless of any values set for `ThreadConfig` or `MaxNoOfExecutionThreads` in `config.ini`; this means that it is not necessary to set either of these parameters.

In addition, NDB 8.0.23 implements a number of new `ndbinfo` information database tables providing information about hardware and CPU availability, as well as CPU usage by NDB data nodes. These tables are listed here:

- `cpudata`
- `cpudata_1sec`
- `cpudata_20sec`
- `cpudata_50ms`
- `cpuinfo`
- `hwinfo`

Some of these tables are not available on every platform supported by NDB Cluster; see the individual descriptions of them for more information.

- **Hierarchical views of NDB database objects.** The `dict_obj_tree` table, added to the `ndbinfo` information database in NDB 8.0.24, can provide hierarchical and tree-like views of many NDB database objects, including the following:
  - Tables and associated indexes
  - Tablespaces and associated data files
  - Logfile groups and associated undo log files

  For more information and examples, see Section 7.14.22, "The ndbinfo dict_obj_tree Table".

- **Index statistics enhancements.** NDB 8.0.24 implements the following improvements in calculation of index statistics:
  - Index statistics were previously collected from one fragment only; this is changed such that this extrapolation is extended to additional fragments.
  - The algorithm used for very small tables, such as those having very few rows where results are discarded, has been improved, so that estimates for such tables should be more accurate than previously.

  For additional information, see Section 7.13, "NDB API Statistics Counters and Variables".
MySQL Cluster Manager 1.4.8 also provides experimental support for NDB Cluster 8.0. MySQL Cluster Manager has an advanced command-line interface that can simplify many complex NDB Cluster management tasks. See MySQL™ Cluster Manager 1.4.8 User Manual, for more information.

### 3.5 Options, Variables, and Parameters Added, Deprecated or Removed in NDB 8.0

- **Parameters Introduced in NDB 8.0**
  - AllowUnresolvedHostNames: When false (default), failure by management node to resolve host name results in fatal error; when true, unresolved host names are reported as warnings only. Added in NDB 8.0.22.
  - AutomaticThreadConfig: Use automatic thread configuration; overrides any settings for ThreadConfig and MaxNoOfExecutionThreads. Added in NDB 8.0.23.
  - ClassicFragmentation: When true, use traditional table fragmentation; set false to enable flexible distribution of table fragments among LDMs. Added in NDB 8.0.23.
  - DiskDataUsingSameDisk: Set to false if Disk Data tablespaces are located on separate physical disks. Added in NDB 8.0.19.
  - MaxDiskDataLatency: Maximum allowed mean latency of disk access (ms) before starting to abort transactions. Added in NDB 8.0.19.
  - NodeGroupTransports: Number of transporters to use between nodes in same node group. Added in NDB 8.0.20.
  - NumCPUs: Specify number of CPUs to use with AutomaticThreadConfig. Added in NDB 8.0.23.
  - PartitionsPerNode: Determines the number of table partitions created on each data node; not used if ClassicFragmentation is enabled. Added in NDB 8.0.23.
  - RequireEncryptedBackup: Whether backups must be encrypted (1 = encryption required, otherwise 0). Added in NDB 8.0.22.
  - ReservedConcurrentIndexOperations: Number of simultaneous index operations having dedicated resources on one data node. Added in NDB 8.0.16.
  - ReservedConcurrentOperations: Number of simultaneous operations having dedicated resources in transaction coordinators on one data node. Added in NDB 8.0.16.
  - ReservedConcurrentScans: Number of simultaneous scans having dedicated resources on one data node. Added in NDB 8.0.16.
Parameters Deprecated in NDB 8.0

- **ReservedConcurrentTransactions**: Number of simultaneous transactions having dedicated resources on one data node. Added in NDB 8.0.16.
- **ReservedFiredTriggers**: Number of triggers having dedicated resources on one data node. Added in NDB 8.0.16.
- **ReservedLocalScans**: Number of simultaneous fragment scans having dedicated resources on one data node. Added in NDB 8.0.16.
- **ReservedTransactionBufferMemory**: Dynamic buffer space (in bytes) for key and attribute data allocated to each data node. Added in NDB 8.0.16.
- **SpinMethod**: Determines spin method used by data node; see documentation for details. Added in NDB 8.0.20.
- **TcpSpinTime**: Time to spin before going to sleep when receiving. Added in NDB 8.0.20.
- **TransactionMemory**: Memory allocated for transactions on each data node. Added in NDB 8.0.19.

Parameters Deprecated in NDB 8.0

The following node configuration parameters have been deprecated in NDB 8.0.

- **BatchSizePerLocalScan**: Used to calculate number of lock records for scan with hold lock. Deprecated in NDB 8.0.19.
- **MaxNoOfConcurrentIndexOperations**: Total number of index operations that can execute simultaneously on one data node. Deprecated in NDB 8.0.19.
- **MaxNoOfConcurrentTransactions**: Maximum number of transactions executing concurrently on this data node, total number of transactions that can be executed concurrently is this value times number of data nodes in cluster. Deprecated in NDB 8.0.19.
- **MaxNoOfFiredTriggers**: Total number of triggers that can fire simultaneously on one data node. Deprecated in NDB 8.0.19.
- **MaxNoOfLocalOperations**: Maximum number of operation records defined on this data node. Deprecated in NDB 8.0.19.
- **MaxNoOfLocalScans**: Maximum number of fragment scans in parallel on this data node. Deprecated in NDB 8.0.19.
- **ReservedTransactionBufferMemory**: Dynamic buffer space (in bytes) for key and attribute data allocated to each data node. Deprecated in NDB 8.0.19.

Parameters Removed in NDB 8.0

No node configuration parameters have been removed in NDB 8.0.

Options and Variables Introduced in NDB 8.0

The following system variables, status variables, and server options have been added in NDB 8.0.

- **Ndb_api_adaptive_send_deferred_count_replica**: Number of adaptive send calls not actually sent by this replica. Added in NDB 8.0.23.
- **Ndb_api_adaptive_send_forced_count_replica**: Number of adaptive sends with forced-send set sent by this replica. Added in NDB 8.0.23.
- **Ndb_api_adaptive_send_unforced_count_replica**: Number of adaptive sends without forced-send sent by this replica. Added in NDB 8.0.23.
- **Ndb_api_bytes_received_count_replica**: Quantity of data (in bytes) received from data nodes by this replica. Added in NDB 8.0.23.
- **Ndb_api_bytes_sent_count_replica**: Quantity of data (in bytes) sent to data nodes by this replica. Added in NDB 8.0.23.

- **Ndb_api_pk_op_count_replica**: Number of operations based on or using primary keys by this replica. Added in NDB 8.0.23.

- **Ndb_api_pruned_scan_count_replica**: Number of scans that have been pruned to one partition by this replica. Added in NDB 8.0.23.

- **Ndb_api_range_scan_count_replica**: Number of range scans that have been started by this replica. Added in NDB 8.0.23.

- **Ndb_api_read_row_count_replica**: Total number of rows that have been read by this replica. Added in NDB 8.0.23.

- **Ndb_api_scan_batch_count_replica**: Number of batches of rows received by this replica. Added in NDB 8.0.23.

- **Ndb_api_table_scan_count_replica**: Number of table scans that have been started, including scans of internal tables, by this replica. Added in NDB 8.0.23.

- **Ndb_api_trans_abort_count_replica**: Number of transactions aborted by this replica. Added in NDB 8.0.23.

- **Ndb_api_trans_close_count_replica**: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this replica. Added in NDB 8.0.23.

- **Ndb_api_trans_commit_count_replica**: Number of transactions committed by this replica. Added in NDB 8.0.23.

- **Ndb_api_trans_local_read_row_count_replica**: Total number of rows that have been read by this replica. Added in NDB 8.0.23.

- **Ndb_api_trans_start_count_replica**: Number of transactions started by this replica. Added in NDB 8.0.23.

- **Ndb_api_uk_op_count_replica**: Number of operations based on or using unique keys by this replica. Added in NDB 8.0.23.

- **Ndb_api_wait_exec_complete_count_replica**: Number of times thread has been blocked while waiting for operation execution to complete by this replica. Added in NDB 8.0.23.

- **Ndb_api_wait_meta_request_count_replica**: Number of times thread has been blocked waiting for metadata-based signal by this replica. Added in NDB 8.0.23.

- **Ndb_api_wait_nanos_count_replica**: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this replica. Added in NDB 8.0.23.

- **Ndb_api_wait_scan_result_count_replica**: Number of times thread has been blocked while waiting for scan-based signal by this replica. Added in NDB 8.0.23.

- **Ndb_config_generation**: Generation number of the current configuration of the cluster. Added in NDB 8.0.24.

- **Ndb_metadata_blacklist_size**: Number of NDB metadata objects that NDB binlog thread has failed to synchronize; renamed in NDB 8.0.22 as Ndb_metadata_excluded_count. Added in NDB 8.0.18.

- **Ndb_metadata_detected_count**: Number of times NDB metadata change monitor thread has detected changes. Added in NDB 8.0.16.

- **Ndb_metadata_excluded_count**: Number of NDB metadata objects that NDB binlog thread has failed to synchronize. Added in NDB 8.0.22.
Options and Variables Deprecated in NDB 8.0

The following system variables, status variables, and options have been deprecated in NDB 8.0.

- `Ndb_api_adaptive_send_deferred_count_slave`: Number of adaptive send calls not actually sent by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_adaptive_send_forced_count_slave`: Number of adaptive sends with forced-send set sent by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_adaptive_send_unforced_count_slave`: Number of adaptive sends without forced-send sent by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_bytes_received_count_slave`: Quantity of data (in bytes) received from data nodes by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_bytes_sent_count_slave`: Quantity of data (in bytes) sent to data nodes by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_pk_op_count_slave`: Number of operations based on or using primary keys by this replica. Deprecated in NDB 8.0.23.
- `Ndb_api_pruned_scan_count_slave`: Number of scans that have been pruned to one partition by this replica. Deprecated in NDB 8.0.23.
Options and Variables Removed in NDB 8.0

The following system variables, status variables, and options have been removed in NDB 8.0.

- Ndb_api_range_scan_count_slave: Number of range scans that have been started by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_read_row_count_slave: Total number of rows that have been read by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_scan_batch_count_slave: Number of batches of rows received by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_table_scan_count_slave: Number of table scans that have been started, including scans of internal tables, by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_trans_abort_count_slave: Number of transactions aborted by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_trans_close_count_slave: Number of transactions aborted (may be greater than sum of TransCommitCount and TransAbortCount) by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_trans_commit_count_slave: Number of transactions committed by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_trans_local_read_row_count_slave: Total number of rows that have been read by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_trans_start_count_slave: Number of transactions started by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_uk_op_count_slave: Number of operations based on or using unique keys by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_wait_exec_complete_count_slave: Number of times thread has been blocked while waiting for operation execution to complete by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_wait_meta_request_count_slave: Number of times thread has been blocked waiting for metadata-based signal by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_wait_nanos_count_slave: Total time (in nanoseconds) spent waiting for some type of signal from data nodes by this replica. Deprecated in NDB 8.0.23.
- Ndb_api_wait_scan_result_count_slave: Number of times thread has been blocked while waiting for scan-based signal by this replica. Deprecated in NDB 8.0.23.
- Ndb_metadata_blacklist_size: Number of NDB metadata objects that NDB binlog thread has failed to synchronize; renamed in NDB 8.0.22 as Ndb_metadata_excluded_count. Removed in NDB 8.0.22.

Options and Variables Removed in NDB 8.0

The following system variables, status variables, and options have been removed in NDB 8.0.

- Ndb_metadata_blacklist_size: Number of NDB metadata objects that NDB binlog thread has failed to synchronize; renamed in NDB 8.0.22 as Ndb_metadata_excluded_count. Removed in NDB 8.0.22.

3.6 MySQL Server Using InnoDB Compared with NDB Cluster
MySQL Server offers a number of choices in storage engines. Since both NDB and InnoDB can serve as transactional MySQL storage engines, users of MySQL Server sometimes become interested in NDB Cluster. They see NDB as a possible alternative or upgrade to the default InnoDB storage engine in MySQL 8.0. While NDB and InnoDB share common characteristics, there are differences in architecture and implementation, so that some existing MySQL Server applications and usage scenarios can be a good fit for NDB Cluster, but not all of them.

In this section, we discuss and compare some characteristics of the NDB storage engine used by NDB 8.0 with InnoDB used in MySQL 8.0. The next few sections provide a technical comparison. In many instances, decisions about when and where to use NDB Cluster must be made on a case-by-case basis, taking all factors into consideration. While it is beyond the scope of this documentation to provide specifics for every conceivable usage scenario, we also attempt to offer some very general guidance on the relative suitability of some common types of applications for NDB as opposed to InnoDB back ends.

NDB Cluster 8.0 uses a mysqld based on MySQL 8.0, including support for InnoDB 1.1. While it is possible to use InnoDB tables with NDB Cluster, such tables are not clustered. It is also not possible to use programs or libraries from an NDB Cluster 8.0 distribution with MySQL Server 8.0, or the reverse.

While it is also true that some types of common business applications can be run either on NDB Cluster or on MySQL Server (most likely using the InnoDB storage engine), there are some important architectural and implementation differences. Section 3.6.1, “Differences Between the NDB and InnoDB Storage Engines”, provides a summary of the these differences. Due to the differences, some usage scenarios are clearly more suitable for one engine or the other; see Section 3.6.2, “NDB and InnoDB Workloads”. This in turn has an impact on the types of applications that better suited for use with NDB or InnoDB. See Section 3.6.3, “NDB and InnoDB Feature Usage Summary”, for a comparison of the relative suitability of each for use in common types of database applications.

For information about the relative characteristics of the NDB and MEMORY storage engines, see When to Use MEMORY or NDB Cluster.

See Alternative Storage Engines, for additional information about MySQL storage engines.

### 3.6.1 Differences Between the NDB and InnoDB Storage Engines

The NDB storage engine is implemented using a distributed, shared-nothing architecture, which causes it to behave differently from InnoDB in a number of ways. For those unaccustomed to working with NDB, unexpected behaviors can arise due to its distributed nature with regard to transactions, foreign keys, table limits, and other characteristics. These are shown in the following table:

<table>
<thead>
<tr>
<th>Feature</th>
<th>InnoDB (MySQL 8.0)</th>
<th>NDB 8.0</th>
</tr>
</thead>
<tbody>
<tr>
<td>MySQL Server Version</td>
<td>8.0</td>
<td>8.0</td>
</tr>
<tr>
<td>InnoDB Version</td>
<td>InnoDB 8.0.25</td>
<td>InnoDB 8.0.25</td>
</tr>
<tr>
<td>NDB Cluster Version</td>
<td>N/A</td>
<td>NDB 8.0.24/8.0.24</td>
</tr>
<tr>
<td>Storage Limits</td>
<td>64TB</td>
<td>128TB</td>
</tr>
<tr>
<td>Foreign Keys</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Transactions</td>
<td>All standard types</td>
<td>READ COMMITTED</td>
</tr>
<tr>
<td>MVCC</td>
<td>Yes</td>
<td>No</td>
</tr>
<tr>
<td>Data Compression</td>
<td>Yes</td>
<td>No (NDB checkpoint and backup files can be compressed)</td>
</tr>
<tr>
<td>Large Row Support (&gt; 14K)</td>
<td>Supported for VARBINARY, VARCHAR, BLOB, and TEXT columns</td>
<td>Supported for BLOB and TEXT columns only (Using these types to store very large amounts of data can lower NDB performance)</td>
</tr>
</tbody>
</table>
### 3.6.2 NDB and InnoDB Workloads

NDB Cluster has a range of unique attributes that make it ideal to serve applications requiring high availability, fast failover, high throughput, and low latency. Due to its distributed architecture and multi-node implementation, NDB Cluster also has specific constraints that may keep some workloads from performing well. A number of major differences in behavior between the NDB and InnoDB storage engines with regard to some common types of database-driven application workloads are shown in the following table:

**Table 3.3 Differences between InnoDB and NDB storage engines, common types of data-driven application workloads.**

<table>
<thead>
<tr>
<th>Workload</th>
<th>InnoDB</th>
<th>NDB Cluster (NDB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>High-Volume OLTP Applications</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>
### 3.6.3 NDB and InnoDB Feature Usage Summary

When comparing application feature requirements to the capabilities of InnoDB with NDB, some are clearly more compatible with one storage engine than the other.

The following table lists supported application features according to the storage engine to which each feature is typically better suited.

<table>
<thead>
<tr>
<th>Workload</th>
<th>InnoDB</th>
<th>NDB Cluster (NDB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>DSS Applications (data marts, analytics)</td>
<td>Yes</td>
<td>Limited (Join operations across OLTP datasets not exceeding 3TB in size)</td>
</tr>
<tr>
<td>Custom Applications</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>Packaged Applications</td>
<td>Yes</td>
<td>Limited (should be mostly primary key access); NDB Cluster 8.0 supports foreign keys</td>
</tr>
<tr>
<td>In-Network Telecoms Applications (HLR, HSS, SDP)</td>
<td>No</td>
<td>Yes</td>
</tr>
<tr>
<td>Session Management and Caching</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>E-Commerce Applications</td>
<td>Yes</td>
<td>Yes</td>
</tr>
<tr>
<td>User Profile Management, AAA Protocol</td>
<td>Yes</td>
<td>Yes</td>
</tr>
</tbody>
</table>

#### 3.7 Known Limitations of NDB Cluster

In the sections that follow, we discuss known limitations in current releases of NDB Cluster as compared with the features available when using the MyISAM and InnoDB storage engines. If you check the “Cluster” category in the MySQL bugs database at [http://bugs.mysql.com](http://bugs.mysql.com), you can find known bugs in the following categories under “MySQL Server:” in the MySQL bugs database at [http://bugs.mysql.com](http://bugs.mysql.com), which we intend to correct in upcoming releases of NDB Cluster:

- NDB Cluster
- Cluster Direct API (NDBAPI)
3.7.1 Noncompliance with SQL Syntax in NDB Cluster

Some SQL statements relating to certain MySQL features produce errors when used with NDB tables, as described in the following list:

- **Temporary tables.** Temporary tables are not supported. Trying either to create a temporary table that uses the NDB storage engine or to alter an existing temporary table to use NDB fails with the error `Table storage engine 'ndbcluster' does not support the create option 'TEMPORARY'`.

- **Indexes and keys in NDB tables.** Keys and indexes on NDB Cluster tables are subject to the following limitations:
  - **Column width.** Attempting to create an index on an NDB table column whose width is greater than 3072 bytes succeeds, but only the first 3072 bytes are actually used for the index. In such cases, a warning `Specified key was too long; max key length is 3072 bytes` is issued, and a `SHOW CREATE TABLE` statement shows the length of the index as 3072.
  - **TEXT and BLOB columns.** You cannot create indexes on NDB table columns that use any of the TEXT or BLOB data types.
  - **FULLTEXT indexes.** The NDB storage engine does not support FULLTEXT indexes, which are possible for MyISAM and InnoDB tables only.

However, you can create indexes on VARCHAR columns of NDB tables.

- **USING HASH keys and NULL.** Using nullable columns in unique keys and primary keys means that queries using these columns are handled as full table scans. To work around this issue, make the column NOT NULL, or re-create the index without the USING HASH option.

- **Prefixes.** There are no prefix indexes; only entire columns can be indexed. (The size of an NDB column index is always the same as the width of the column in bytes, up to and including 3072 bytes, as described earlier in this section. Also see Section 3.7.6, "Unsupported or Missing Features in NDB Cluster", for additional information.)

- **BIT columns.** A BIT column cannot be a primary key, unique key, or index, nor can it be part of a composite primary key, unique key, or index.

- **AUTO_INCREMENT columns.** Like other MySQL storage engines, the NDB storage engine can handle a maximum of one AUTO_INCREMENT column per table. However, in the case of an NDB table with no explicit primary key, an AUTO_INCREMENT column is automatically defined and used as a "hidden" primary key. For this reason, you cannot define a table that has an explicit AUTO_INCREMENT column unless that column is also declared using the PRIMARY KEY option.
Noncompliance with SQL Syntax in NDB Cluster

Attempting to create a table with an AUTO_INCREMENT column that is not the table's primary key, and using the NDB storage engine, fails with an error.

- **Restrictions on foreign keys.** Support for foreign key constraints in NDB 8.0 is comparable to that provided by InnoDB, subject to the following restrictions:
  - Every column referenced as a foreign key requires an explicit unique key, if it is not the table's primary key.
  - **ON UPDATE CASCADE** is not supported when the reference is to the parent table's primary key. This is because an update of a primary key is implemented as a delete of the old row (containing the old primary key) plus an insert of the new row (with a new primary key). This is not visible to the NDB kernel, which views these two rows as being the same, and thus has no way of knowing that this update should be cascaded.
  - As of NDB 8.0.16: **ON DELETE CASCADE** is not supported where the child table contains one or more columns of any of the TEXT or BLOB types. (Bug #89511, Bug #27484882)
  - **SET DEFAULT** is not supported. (Also not supported by InnoDB.)
  - The **NO ACTION** keyword is accepted but treated as **RESTRICT.NO ACTION**, which is a standard SQL keyword, is the default in MySQL 8.0. (Also the same as with InnoDB.)
  - In earlier versions of NDB Cluster, when creating a table with foreign key referencing an index in another table, it sometimes appeared possible to create the foreign key even if the order of the columns in the indexes did not match, due to the fact that an appropriate error was not always returned internally. A partial fix for this issue improved the error used internally to work in most cases; however, it remains possible for this situation to occur in the event that the parent index is a unique index. (Bug #18094360)

For more information, see FOREIGN KEY Constraints, and FOREIGN KEY Constraints.

- **NDB Cluster and geometry data types.**
  Geometry data types (WKT and WKB) are supported for NDB tables. However, spatial indexes are not supported.

- **Character sets and binary log files.** Currently, the ndb_apply_status and ndb_binlog_index tables are created using the latin1 (ASCII) character set. Because names of binary logs are recorded in this table, binary log files named using non-Latin characters are not referenced correctly in these tables. This is a known issue, which we are working to fix. (Bug #50226)

To work around this problem, use only Latin-1 characters when naming binary log files or setting any of the --basedir, --log-bin, or --log-bin-index options.

- **Creating NDB tables with user-defined partitioning.** Support for user-defined partitioning in NDB Cluster is restricted to [LINEAR] KEY partitioning. Using any other partitioning type with ENGINE=NDB or ENGINE=NDBCLUSTER in a CREATE TABLE statement results in an error.

  It is possible to override this restriction, but doing so is not supported for use in production settings. For details, see User-defined partitioning and the NDB storage engine (NDB Cluster).

**Default partitioning scheme.** All NDB Cluster tables are by default partitioned by KEY using the table’s primary key as the partitioning key. If no primary key is explicitly set for the table, the “hidden” primary key automatically created by the NDB storage engine is used instead. For additional discussion of these and related issues, see KEY Partitioning.

CREATE TABLE and ALTER TABLE statements that would cause a user-partitioned NDBCLUSTER table not to meet either or both of the following two requirements are not permitted, and fail with an error:
1. The table must have an explicit primary key.

2. All columns listed in the table’s partitioning expression must be part of the primary key.

**Exception.** If a user-partitioned NDBCLUSTER table is created using an empty column-list (that is, using `PARTITION BY [LINEAR] KEY()`), then no explicit primary key is required.

**Maximum number of partitions for NDBCLUSTER tables.** The maximum number of partitions that can defined for a NDBCLUSTER table when employing user-defined partitioning is 8 per node group. (See Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”, for more information about NDB Cluster node groups.

**DROP PARTITION not supported.** It is not possible to drop partitions from NDB tables using `ALTER TABLE ... DROP PARTITION`. The other partitioning extensions to `ALTER TABLE—ADD PARTITION, REORGANIZE PARTITION, and COALESCE PARTITION—are supported for NDB tables, but use copying and so are not optimized. See Management of RANGE and LIST Partitions and `ALTER TABLE Statement`.

- **JSON data type.** The MySQL JSON data type is supported for NDB tables in the mysqlld supplied with NDB 8.0.

  An NDB table can have a maximum of 3 JSON columns.

  The NDB API has no special provision for working with JSON data, which it views simply as BLOB data. Handling data as JSON must be performed by the application.

### 3.7.2 Limits and Differences of NDB Cluster from Standard MySQL Limits

In this section, we list limits found in NDB Cluster that either differ from limits found in, or that are not found in, standard MySQL.

**Memory usage and recovery.** Memory consumed when data is inserted into an NDB table is not automatically recovered when deleted, as it is with other storage engines. Instead, the following rules hold true:

- A **DELETE** statement on an NDB table makes the memory formerly used by the deleted rows available for re-use by inserts on the same table only. However, this memory can be made available for general re-use by performing `OPTIMIZE TABLE`.

  A rolling restart of the cluster also frees any memory used by deleted rows. See Section 7.5, “Performing a Rolling Restart of an NDB Cluster”.

- A **DROP TABLE** or **TRUNCATE TABLE** operation on an NDB table frees the memory that was used by this table for re-use by any NDB table, either by the same table or by another NDB table.

  **Note**
  
  Recall that **TRUNCATE TABLE** drops and re-creates the table. See **TRUNCATE TABLE Statement**.

- **Limits imposed by the cluster’s configuration.** A number of hard limits exist which are configurable, but available main memory in the cluster sets limits. See the complete list of configuration parameters in Section 5.3, “NDB Cluster Configuration Files”. Most configuration parameters can be upgraded online. These hard limits include:

  - **Database memory size and index memory size** (`DataMemory` and `IndexMemory`, respectively).

    `DataMemory` is allocated as 32KB pages. As each `DataMemory` page is used, it is assigned to a specific table; once allocated, this memory cannot be freed except by dropping the table.

    See Section 5.3.6, “Defining NDB Cluster Data Nodes”, for more information.
The maximum number of operations that can be performed per transaction is set using the configuration parameters `MaxNoOfConcurrentOperations` and `MaxNoOfLocalOperations`.

**Note**

Bulk loading, `TRUNCATE TABLE`, and `ALTER TABLE` are handled as special cases by running multiple transactions, and so are not subject to this limitation.

Different limits related to tables and indexes. For example, the maximum number of ordered indexes in the cluster is determined by `MaxNoOfOrderedIndexes`, and the maximum number of ordered indexes per table is 16.

**Node and data object maximums.** The following limits apply to numbers of cluster nodes and metadata objects:

- As of NDB 8.0.18, the maximum number of data nodes is 145. (Previously, this was 48.)
  A data node must have a node ID in the range of 1 to 144, inclusive. (In NDB 8.0.17 and earlier releases, this was 1 to 48, inclusive.)
  Management and API nodes may use node IDs in the range 1 to 255, inclusive.
- The total maximum number of nodes in an NDB Cluster is 255. This number includes all SQL nodes (MySQL Servers), API nodes (applications accessing the cluster other than MySQL servers), data nodes, and management servers.
- The maximum number of metadata objects in current versions of NDB Cluster is 20320. This limit is hard-coded.

See Section 3.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”, for more information.

### 3.7.3 Limits Relating to Transaction Handling in NDB Cluster

A number of limitations exist in NDB Cluster with regard to the handling of transactions. These include the following:

**Transaction isolation level.** The `NDBCLUSTER` storage engine supports only the `READ COMMITTED` transaction isolation level. (InnoDB, for example, supports `READ COMMITTED`, `READ UNCOMMITTED`, `REPEATABLE READ`, and `SERIALIZABLE`.) You should keep in mind that NDB implements `READ COMMITTED` on a per-row basis; when a read request arrives at the data node storing the row, what is returned is the last committed version of the row at that time.

Uncommitted data is never returned, but when a transaction modifying a number of rows commits concurrently with a transaction reading the same rows, the transaction performing the read can observe “before” values, “after” values, or both, for different rows among these, due to the fact that a given row read request can be processed either before or after the commit of the other transaction.

To ensure that a given transaction reads only before or after values, you can impose row locks using `SELECT ... LOCK IN SHARE MODE`. In such cases, the lock is held until the owning transaction is committed. Using row locks can also cause the following issues:

- Increased frequency of lock wait timeout errors, and reduced concurrency
- Increased transaction processing overhead due to reads requiring a commit phase
- Possibility of exhausting the available number of concurrent locks, which is limited by `MaxNoOfConcurrentOperations`
NDB uses **READ COMMITTED** for all reads unless a modifier such as **LOCK IN SHARE MODE** or **FOR UPDATE** is used. **LOCK IN SHARE MODE** causes shared row locks to be used; **FOR UPDATE** causes exclusive row locks to be used. Unique key reads have their locks upgraded automatically by NDB to ensure a self-consistent read; **BLOB** reads also employ extra locking for consistency.

See Section 7.8.4, “NDB Cluster Backup Troubleshooting”, for information on how NDB Cluster’s implementation of transaction isolation level can affect backup and restoration of **NDB** databases.

- **Transactions and **BLOB** or **TEXT** columns.** **NDBCLUSTER** stores only part of a column value that uses any of MySQL’s **BLOB** or **TEXT** data types in the table visible to MySQL; the remainder of the **BLOB** or **TEXT** is stored in a separate internal table that is not accessible to MySQL. This gives rise to two related issues of which you should be aware whenever executing **SELECT** statements on tables that contain columns of these types:

1. For any **SELECT** from an NDB Cluster table: If the **SELECT** includes a **BLOB** or **TEXT** column, the **READ COMMITTED** transaction isolation level is converted to a read with read lock. This is done to guarantee consistency.

2. For any **SELECT** which uses a unique key lookup to retrieve any columns that use any of the **BLOB** or **TEXT** data types and that is executed within a transaction, a shared read lock is held on the table for the duration of the transaction—that is, until the transaction is either committed or aborted.

This issue does not occur for queries that use index or table scans, even against **NDB** tables having **BLOB** or **TEXT** columns.

For example, consider the table t defined by the following **CREATE TABLE** statement:

```sql
CREATE TABLE t (
    a INT NOT NULL AUTO_INCREMENT PRIMARY KEY,
    b INT NOT NULL,
    c INT NOT NULL,
    d TEXT,
    INDEX i(b),
    UNIQUE KEY u(c)
) ENGINE = NDB,
```

The following query on t causes a shared read lock, because it uses a unique key lookup:

```sql
SELECT * FROM t WHERE c = 1;
```

However, none of the four queries shown here causes a shared read lock:

```sql
SELECT * FROM t WHERE b = 1;
SELECT * FROM t WHERE d = '1';
SELECT * FROM t;
SELECT b,c WHERE a = 1;
```

This is because, of these four queries, the first uses an index scan, the second and third use table scans, and the fourth, while using a primary key lookup, does not retrieve the value of any **BLOB** or **TEXT** columns.

You can help minimize issues with shared read locks by avoiding queries that use unique key lookups that retrieve **BLOB** or **TEXT** columns, or, in cases where such queries are not avoidable, by committing transactions as soon as possible afterward.

- **Unique key lookups and transaction isolation.** Unique indexes are implemented in **NDB** using a hidden index table which is maintained internally. When a user-created **NDB** table is accessed using a unique index, the hidden index table is first read to find the primary key that is then used to read the user-created table. To avoid modification of the index during this double-read operation, the row found in the hidden index table is locked. When a row referenced by a unique index in the user-created **NDB** table is updated, the hidden index table is subject to an exclusive lock by the transaction.
Limits Relating to Transaction Handling in NDB Cluster

in which the update is performed. This means that any read operation on the same (user-created) NDB table must wait for the update to complete. This is true even when the transaction level of the read operation is **READ COMMITTED**.

One workaround which can be used to bypass potentially blocking reads is to force the SQL node to ignore the unique index when performing the read. This can be done by using the `IGNORE INDEX` index hint as part of the `SELECT` statement reading the table (see Index Hints). Because the MySQL server creates a shadowing ordered index for every unique index created in NDB, this lets the ordered index be read instead, and avoids unique index access locking. The resulting read is as consistent as a committed read by primary key, returning the last committed value at the time the row is read.

Reading via an ordered index makes less efficient use of resources in the cluster, and may have higher latency.

It is also possible to avoid using the unique index for access by querying for ranges rather than for unique values.

• **Rollbacks.** There are no partial transactions, and no partial rollbacks of transactions. A duplicate key or similar error causes the entire transaction to be rolled back.

This behavior differs from that of other transactional storage engines such as InnoDB that may roll back individual statements.

• **Transactions and memory usage.**

As noted elsewhere in this chapter, NDB Cluster does not handle large transactions well; it is better to perform a number of small transactions with a few operations each than to attempt a single large transaction containing a great many operations. Among other considerations, large transactions require very large amounts of memory. Because of this, the transactional behavior of a number of MySQL statements is affected as described in the following list:

• `TRUNCATE TABLE` is not transactional when used on NDB tables. If a `TRUNCATE TABLE` fails to empty the table, then it must be re-run until it is successful.

• `DELETE FROM` (even with no `WHERE` clause) *is* transactional. For tables containing a great many rows, you may find that performance is improved by using several `DELETE FROM ... LIMIT ...` statements to “chunk” the delete operation. If your objective is to empty the table, then you may wish to use `TRUNCATE TABLE` instead.

• **LOAD DATA statements.** `LOAD DATA` is not transactional when used on NDB tables.

```
Important
When executing a `LOAD DATA` statement, the NDB engine performs commits at irregular intervals that enable better utilization of the communication network. It is not possible to know ahead of time when such commits take place.
```

• **ALTER TABLE and transactions.** When copying an NDB table as part of an `ALTER TABLE`, the creation of the copy is nontransactional. (In any case, this operation is rolled back when the copy is deleted.)

• **Transactions and the COUNT() function.** When using NDB Cluster Replication, it is not possible to guarantee the transactional consistency of the `COUNT()` function on the replica. In other words, when performing on the source a series of statements (INSERT, DELETE, or both) that changes the number of rows in a table within a single transaction, executing `SELECT COUNT(*) FROM table` queries on the replica may yield intermediate results. This is due to the fact that `SELECT COUNT(*)` may perform dirty reads, and is not a bug in the NDB storage engine. (See Bug #31321 for more information.)
3.7.4 NDB Cluster Error Handling

Starting, stopping, or restarting a node may give rise to temporary errors causing some transactions to fail. These include the following cases:

- **Temporary errors.** When first starting a node, it is possible that you may see Error 1204 `Temporary failure, distribution changed` and similar temporary errors.

- **Errors due to node failure.** The stopping or failure of any data node can result in a number of different node failure errors. (However, there should be no aborted transactions when performing a planned shutdown of the cluster.)

In either of these cases, any errors that are generated must be handled within the application. This should be done by retrying the transaction.

See also Section 3.7.2, “Limits and Differences of NDB Cluster from Standard MySQL Limits”.

3.7.5 Limits Associated with Database Objects in NDB Cluster

Some database objects such as tables and indexes have different limitations when using the NDBCLUSTER storage engine:

- **Number of database objects.** The maximum number of all NDB database objects in a single NDB Cluster—including databases, tables, and indexes—is limited to 20320.

- **Attributes per table.** The maximum number of attributes (that is, columns and indexes) that can belong to a given table is 512.

- **Attributes per key.** The maximum number of attributes per key is 32.

- **Row size.** Beginning with NDB 8.0.18, the maximum permitted size of any one row is 30000 bytes, in NDB 8.0.17 and earlier releases, this limit is 14000 bytes.

  Each BLOB or TEXT column contributes $256 + 8 = 264$ bytes to this total; this includes JSON columns. See String Type Storage Requirements, as well as JSON Storage Requirements, for more information relating to these types.

  In addition, the maximum offset for a fixed-width column of an NDB table is 8188 bytes; attempting to create a table that violates this limitation fails with NDB error 851 `Maximum offset for fixed-size columns exceeded`. For memory-based columns, you can work around this limitation by using a variable-width column type such as VARCHAR or defining the column as `COLUMN_FORMAT=DYNAMIC`; this does not work with columns stored on disk. For disk-based columns, you may be able to do so by reordering one or more of the table’s disk-based columns such that the combined width of all but the disk-based column defined last in the `CREATE TABLE` statement used to create the table does not exceed 8188 bytes, less any possible rounding performed for some data types such as CHAR or VARCHAR; otherwise it is necessary to use memory-based storage for one or more of the offending column or columns instead.

- **BIT column storage per table.** The maximum combined width for all BIT columns used in a given NDB table is 4096.

- **FIXED column storage.** NDB Cluster 8.0 supports a maximum of 128 TB per fragment of data in FIXED columns.

3.7.6 Unsupported or Missing Features in NDB Cluster

A number of features supported by other storage engines are not supported for NDB tables. Trying to use any of these features in NDB Cluster does not cause errors in or of itself; however, errors may occur in applications that expects the features to be supported or enforced. Statements referencing such features, even if effectively ignored by NDB, must be syntactically and otherwise valid.
3.7.7 Limitations Relating to Performance in NDB Cluster

The following performance issues are specific to or especially pronounced in NDB Cluster:

- **Range scans.** There are query performance issues due to sequential access to the NDB storage engine; it is also relatively more expensive to do many range scans than it is with either MyISAM or InnoDB.

- **Reliability of Records in range.** The Records in range statistic is available but is not completely tested or officially supported. This may result in nonoptimal query plans in some cases. If necessary, you can employ USE INDEX or FORCE INDEX to alter the execution plan. See Index Hints, for more information on how to do this.

### Limitations Relating to Performance in NDB Cluster

#### Index prefixes.

Prefixes on indexes are not supported for NDB tables. If a prefix is used as part of an index specification in a statement such as CREATE TABLE, ALTER TABLE, or CREATE INDEX, the prefix is not created by NDB.

A statement containing an index prefix, and creating or modifying an NDB table, must still be syntactically valid. For example, the following statement always fails with Error 1089 Incorrect prefix key; the used key part isn't a string, the used length is longer than the key part, or the storage engine doesn't support unique prefix keys, regardless of storage engine:

```sql
CREATE TABLE t1 (  
c1 INT NOT NULL,  
c2 VARCHAR(100),  
INDEX i1 (c2(500))  
);
```

This happens on account of the SQL syntax rule that no index may have a prefix larger than itself.

#### Savepoints and rollbacks.

Savepoints and rollbacks to savepoints are ignored as in MyISAM.

#### Durability of commits.

There are no durable commits on disk. Commits are replicated, but there is no guarantee that logs are flushed to disk on commit.

#### Replication.

Statement-based replication is not supported. Use --binlog-format=ROW (or --binlog-format=MIXED) when setting up cluster replication. See Chapter 8, NDB Cluster Replication, for more information.

Replication using global transaction identifiers (GTIDs) is not compatible with NDB Cluster, and is not supported in NDB Cluster 8.0. Do not enable GTIDs when using the NDB storage engine, as this is very likely to cause problems up to and including failure of NDB Cluster Replication.

Semisynchronous replication is not supported in NDB Cluster.

#### Generated columns.

The NDB storage engine does not support indexes on virtual generated columns.

As with other storage engines, you can create an index on a stored generated column, but you should bear in mind that NDB uses DataMemory for storage of the generated column as well as IndexMemory for the index. See JSON columns and indirect indexing in NDB Cluster, for an example.

NDB Cluster writes changes in stored generated columns to the binary log, but does log not those made to virtual columns. This should not effect NDB Cluster Replication or replication between NDB and other MySQL storage engines.

**Note**

See Section 3.7.3, “Limits Relating to Transaction Handling in NDB Cluster”, for more information relating to limitations on transaction handling in NDB.
• **Unique hash indexes.**  Unique hash indexes created with `USING HASH` cannot be used for accessing a table if `NULL` is given as part of the key.

### 3.7.8 Issues Exclusive to NDB Cluster

The following are limitations specific to the **NDB** storage engine:

- **Machine architecture.**  All machines used in the cluster must have the same architecture. That is, all machines hosting nodes must be either big-endian or little-endian, and you cannot use a mixture of both. For example, you cannot have a management node running on a PowerPC which directs a data node that is running on an x86 machine. This restriction does not apply to machines simply running `mysql` or other clients that may be accessing the cluster's SQL nodes.

- **Binary logging.**  NDB Cluster has the following limitations or restrictions with regard to binary logging:
  
  - `sql_log_bin` has no effect on data operations; however, it is supported for schema operations.
  
  - NDB Cluster cannot produce a binary log for tables having `BLOB` columns but no primary key.
  
  - Only the following schema operations are logged in a cluster binary log which is *not* on the `mysqld` executing the statement:
    
    - `CREATE TABLE`
    - `ALTER TABLE`
    - `DROP TABLE`
    - `CREATE DATABASE / CREATE SCHEMA`
    - `DROP DATABASE / DROP SCHEMA`
    - `CREATE TABLESPACE`
    - `ALTER TABLESPACE`
    - `DROP TABLESPACE`
    - `CREATE LOGFILE GROUP`
    - `ALTER LOGFILE GROUP`
    - `DROP LOGFILE GROUP`

- **Schema operations.**  Schema operations (DDL statements) are rejected while any data node restarts. Schema operations are also not supported while performing an online upgrade or downgrade.

- **Number of fragment replicas.**  The number of fragment replicas, as determined by the `NoOfReplicas` data node configuration parameter, is the number of copies of all data stored by NDB Cluster. Setting this parameter to 1 means there is only a single copy; in this case, no redundancy is provided, and the loss of a data node entails loss of data. To guarantee redundancy, and thus preservation of data even if a data node fails, set this parameter to 2, which is the default and recommended value in production.

  Setting `NoOfReplicas` to a value greater than 2 is supported (to a maximum of 4) but unnecessary to guard against loss of data.

  See also Section 3.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

### 3.7.9 Limitations Relating to NDB Cluster Disk Data Storage
Disk Data object maximums and minimums. Disk data objects are subject to the following maximums and minimums:

- Maximum number of tablespaces: $2^{32}$ (4294967296)
- Maximum number of data files per tablespace: $2^{16}$ (65536)
- The minimum and maximum possible sizes of extents for tablespace data files are 32K and 2G, respectively. See CREATE TABLESPACE Statement, for more information.

In addition, when working with NDB Disk Data tables, you should be aware of the following issues regarding data files and extents:

- Data files use DataMemory. Usage is the same as for in-memory data.
- Data files use file descriptors. It is important to keep in mind that data files are always open, which means the file descriptors are always in use and cannot be re-used for other system tasks.
- Extents require sufficient DiskPageBufferMemory; you must reserve enough for this parameter to account for all memory used by all extents (number of extents times size of extents).

Disk Data tables and diskless mode. Use of Disk Data tables is not supported when running the cluster in diskless mode.

3.7.10 Limitations Relating to Multiple NDB Cluster Nodes

Multiple SQL nodes. The following are issues relating to the use of multiple MySQL servers as NDB Cluster SQL nodes, and are specific to the NDBCLUSTER storage engine:

- Stored programs not distributed. Stored procedures, stored functions, triggers, and scheduled events are all supported by tables using the NDB storage engine, but these do not propagate automatically between MySQL Servers acting as Cluster SQL nodes, and must be re-created separately on each SQL node. See Stored routines and triggers in NDB Cluster.

- No distributed table locks. A LOCK TABLES works only for the SQL node on which the lock is issued; no other SQL node in the cluster “sees” this lock. This is also true for a lock issued by any statement that locks tables as part of its operations. (See next item for an example.)

- ALTER TABLE operations. ALTER TABLE is not fully locking when running multiple MySQL servers (SQL nodes). (As discussed in the previous item, NDB Cluster does not support distributed table locks.)

Multiple management nodes.

When using multiple management servers:

- If any of the management servers are running on the same host, you must give nodes explicit IDs in connection strings because automatic allocation of node IDs does not work across multiple management servers on the same host. This is not required if every management server resides on a different host.

- When a management server starts, it first checks for any other management server in the same NDB Cluster, and upon successful connection to the other management server uses its configuration data. This means that the management server --reload and --initial startup options are ignored unless the management server is the only one running. It also means that, when performing a rolling restart of an NDB Cluster with multiple management nodes, the management server reads its own configuration file if (and only if) it is the only management server running in this NDB Cluster. See Section 7.5, “Performing a Rolling Restart of an NDB Cluster”, for more information.

Multiple network addresses. Multiple network addresses per data node are not supported. Use of these is liable to cause problems: In the event of a data node failure, an SQL node waits for
confirmation that the data node went down but never receives it because another route to that data node remains open. This can effectively make the cluster inoperable.

**Note**

It is possible to use multiple network hardware interfaces (such as Ethernet cards) for a single data node, but these must be bound to the same address. This also means that it is not possible to use more than one [tcp] section per connection in the `config.ini` file. See Section 5.3.10, “NDB Cluster TCP/IP Connections”, for more information.

### 3.7.11 Previous NDB Cluster Issues Resolved in NDB Cluster 8.0

A number of limitations and related issues that existed in earlier versions of NDB Cluster have been resolved in NDB 8.0. These are described briefly in the following list:

- **Database and table names.** Prior to NDB 8.0.18, when using the NDB storage engine, the maximum allowed length both for database names and for table names is 63 characters, and a statement using a database name or table name longer than this limit failed with an appropriate error. As of NDB 8.0.18, this restriction is lifted and identifiers for NDB databases and tables may now use up to 64 bytes, as with other MySQL database and table names.

- **IPv6 support.** Prior to NDB 8.0.22, it was necessary for all network addresses used for connections between nodes within an NDB Cluster to use or to be resolvable to IPv4 addresses. Beginning with NDB 8.0.22, NDB supports IPv6 addresses for all types of cluster nodes, as well as for applications that use the NDB API or MGM API.
This section describes the basics for planning, installing, configuring, and running an NDB Cluster. Whereas the examples in Chapter 5, Configuration of NDB Cluster provide more in-depth information on a variety of clustering options and configuration, the result of following the guidelines and procedures outlined here should be a usable NDB Cluster which meets the minimum requirements for availability and safeguarding of data.

For information about upgrading or downgrading an NDB Cluster between release versions, see Section 4.7, “Upgrading and Downgrading NDB Cluster”.

This section covers hardware and software requirements; networking issues; installation of NDB Cluster; basic configuration issues; starting, stopping, and restarting the cluster; loading of a sample database; and performing queries.

NDB Cluster also provides the NDB Cluster Auto-Installer (now deprecated), a web-based graphical installer, as part of the NDB Cluster distribution. The Auto-Installer can be used to perform basic installation and setup of an NDB Cluster on one (for testing) or more host computers. See Section 4.8, “The NDB Cluster Auto-Installer (No longer supported)”, for more information.

Assumptions. The following sections make a number of assumptions regarding the cluster’s physical and network configuration. These assumptions are discussed in the next few paragraphs.

Cluster nodes and host computers. The cluster consists of four nodes, each on a separate host computer, and each with a fixed network address on a typical Ethernet network as shown here:

Table 4.1 Network addresses of nodes in example cluster

<table>
<thead>
<tr>
<th>Node</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management node (<strong>mgmd</strong>)</td>
<td>198.51.100.10</td>
</tr>
<tr>
<td>SQL node (<strong>mysqld</strong>)</td>
<td>198.51.100.20</td>
</tr>
<tr>
<td>Data node &quot;A&quot; (<strong>ndbd</strong>)</td>
<td>198.51.100.30</td>
</tr>
</tbody>
</table>
Node IP Address
Data node "B" (ndbd) 198.51.100.40

This setup is also shown in the following diagram:

Figure 4.1 NDB Cluster Multi-Computer Setup

Network addressing. In the interest of simplicity (and reliability), this How-To uses only numeric IP addresses. However, if DNS resolution is available on your network, it is possible to use host names in lieu of IP addresses in configuring Cluster. Alternatively, you can use the `hosts` file (typically `/etc/hosts` for Linux and other Unix-like operating systems, `C:\WINDOWS\system32\drivers\etc\hosts` on Windows, or your operating system's equivalent) for providing a means to do host lookup if such is available.

Prior to NDB 8.0.22, all network addresses used for connections to or from data and management nodes must use or be resolvable using IPv4. This includes addresses used by SQL nodes to contact the other nodes. Beginning with NDB 8.0.22, NDB Cluster supports IPv6 for connections between any and all cluster nodes.

Potential hosts file issues. A common problem when trying to use host names for Cluster nodes arises because of the way in which some operating systems (including some Linux distributions) set up the system's own host name in the `/etc/hosts` during installation. Consider two machines with the host names `ndb1` and `ndb2`, both in the `cluster` network domain. Red Hat Linux (including some derivatives such as CentOS and Fedora) places the following entries in these machines' `/etc/hosts` files:

```plaintext
# ndb1 /etc/hosts:
127.0.0.1   ndb1.cluster ndb1 localhost.localdomain localhost

# ndb2 /etc/hosts:
127.0.0.1   ndb2.cluster ndb2 localhost.localdomain localhost
```

SUSE Linux (including OpenSUSE) places these entries in the machines' `/etc/hosts` files:

```plaintext
# ndb1 /etc/hosts:
127.0.0.1   localhost
127.0.0.2   ndb1.cluster ndb1

# ndb2 /etc/hosts:
127.0.0.1   localhost
```
In both instances, ndb1 routes ndb1.cluster to a loopback IP address, but gets a public IP address from DNS for ndb2.cluster, while ndb2 routes ndb2.cluster to a loopback address and obtains a public address for ndb1.cluster. The result is that each data node connects to the management server, but cannot tell when any other data nodes have connected, and so the data nodes appear to hang while starting.

**Caution**

You cannot mix localhost and other host names or IP addresses in config.ini. For these reasons, the solution in such cases (other than to use IP addresses for all config.ini HostName entries) is to remove the fully qualified host names from /etc/hosts and use these in config.ini for all cluster hosts.

**Host computer type.** Each host computer in our installation scenario is an Intel-based desktop PC running a supported operating system installed to disk in a standard configuration, and running no unnecessary services. The core operating system with standard TCP/IP networking capabilities should be sufficient. Also for the sake of simplicity, we also assume that the file systems on all hosts are set up identically. In the event that they are not, you should adapt these instructions accordingly.

**Network hardware.** Standard 100 Mbps or 1 gigabit Ethernet cards are installed on each machine, along with the proper drivers for the cards, and that all four hosts are connected through a standard-issue Ethernet networking appliance such as a switch. (All machines should use network cards with the same throughput. That is, all four machines in the cluster should have 100 Mbps cards or all four machines should have 1 Gbps cards.) NDB Cluster works in a 100 Mbps network; however, gigabit Ethernet provides better performance.

**Important**

NDB Cluster is not intended for use in a network for which throughput is less than 100 Mbps or which experiences a high degree of latency. For this reason (among others), attempting to run an NDB Cluster over a wide area network such as the Internet is not likely to be successful, and is not supported in production.

**Sample data.** We use the world database which is available for download from the MySQL website (see https://dev.mysql.com/doc/index-other.html). We assume that each machine has sufficient memory for running the operating system, required NDB Cluster processes, and (on the data nodes) storing the database.

For general information about installing MySQL, see Installing and Upgrading MySQL. For information about installation of NDB Cluster on Linux and other Unix-like operating systems, see Section 4.1, “Installation of NDB Cluster on Linux”. For information about installation of NDB Cluster on Windows operating systems, see Section 4.2, “Installing NDB Cluster on Windows”.

For general information about NDB Cluster hardware, software, and networking requirements, see Section 3.3, “NDB Cluster Hardware, Software, and Networking Requirements”.

### 4.1 Installation of NDB Cluster on Linux

This section covers installation methods for NDB Cluster on Linux and other Unix-like operating systems. While the next few sections refer to a Linux operating system, the instructions and procedures given there should be easily adaptable to other supported Unix-like platforms. For manual installation and setup instructions specific to Windows systems, see Section 4.2, “Installing NDB Cluster on Windows”.

Each NDB Cluster host computer must have the correct executable programs installed. A host running an SQL node must have installed on it a MySQL Server binary (mysqld). Management nodes require
Installing an NDB Cluster Binary Release on Linux

the management server daemon (ndb_mgmd); data nodes require the data node daemon (ndbd or ndbmd). It is not necessary to install the MySQL Server binary on management node hosts and data node hosts. It is recommended that you also install the management client (ndb_mgm) on the management server host.

Installation of NDB Cluster on Linux can be done using precompiled binaries from Oracle (downloaded as a .tar.gz archive), with RPM packages (also available from Oracle), or from source code. All three of these installation methods are described in the section that follow.

Regardless of the method used, it is still necessary following installation of the NDB Cluster binaries to create configuration files for all cluster nodes, before you can start the cluster. See Section 4.3, “Initial Configuration of NDB Cluster”.

4.1.1 Installing an NDB Cluster Binary Release on Linux

This section covers the steps necessary to install the correct executables for each type of Cluster node from precompiled binaries supplied by Oracle.

For setting up a cluster using precompiled binaries, the first step in the installation process for each cluster host is to download the binary archive from the NDB Cluster downloads page. (For the most recent 64-bit NDB 8.0 release, this is mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64.tar.gz.) We assume that you have placed this file in each machine’s /var/tmp directory.

If you require a custom binary, see Installing MySQL Using a Development Source Tree.

```
Note
After completing the installation, do not yet start any of the binaries. We show you how to do so following the configuration of the nodes (see Section 4.3, “Initial Configuration of NDB Cluster”).
```

**SQL nodes.** On each of the machines designated to host SQL nodes, perform the following steps as the system root user:

1. Check your /etc/passwd and /etc/group files (or use whatever tools are provided by your operating system for managing users and groups) to see whether there is already a mysql group and mysql user on the system. Some OS distributions create these as part of the operating system installation process. If they are not already present, create a new mysql user group, and then add a mysql user to this group:

```
shell> groupadd mysql
shell> useradd -g mysql -s /bin/false mysql
```

The syntax for useradd and groupadd may differ slightly on different versions of Unix, or they may have different names such as adduser and addgroup.

2. Change location to the directory containing the downloaded file, unpack the archive, and create a symbolic link named mysql to the mysql directory.

```
shell> cd /var/tmp
shell> tar -C /usr/local -xzvf mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64.tar.gz
shell> ln -s /usr/local/mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64 /usr/local/mysql
```

3. Change location to the mysql directory and set up the system databases using mysqld --initialize as shown here:

```
shell> cd mysql
```
Installing an NDB Cluster Binary Release on Linux

shell> mysqld --initialize

This generates a random password for the MySQL root account. If you do not want the random password to be generated, you can substitute the --initialize-insecure option for --initialize. In either case, you should review Initializing the Data Directory, for additional information before performing this step. See also mysql_secure_installation — Improve MySQL Installation Security.

4. Set the necessary permissions for the MySQL server and data directories:

shell> chown -R root .
shell> chown -R mysql data
shell> chgrp -R mysql .

5. Copy the MySQL startup script to the appropriate directory, make it executable, and set it to start when the operating system is booted up:

shell> cp support-files/mysql.server /etc/rc.d/init.d/
shell> chmod +x /etc/rc.d/init.d/mysql.server
shell> chkconfig --add mysql.server

(The startup scripts directory may vary depending on your operating system and version—for example, in some Linux distributions, it is /etc/init.d.)

Here we use Red Hat's chkconfig for creating links to the startup scripts; use whatever means is appropriate for this purpose on your platform, such as update-rc.d on Debian.

Remember that the preceding steps must be repeated on each machine where an SQL node is to reside.

Data nodes. Installation of the data nodes does not require the mysqld binary. Only the NDB Cluster data node executable ndbd (single-threaded) or ndbmtd (multithreaded) is required. These binaries can also be found in the .tar.gz archive. Again, we assume that you have placed this archive in /var/tmp.

As system root (that is, after using sudo, su root, or your system's equivalent for temporarily assuming the system administrator account's privileges), perform the following steps to install the data node binaries on the data node hosts:

1. Change location to the /var/tmp directory, and extract the ndbd and ndbmtd binaries from the archive into a suitable directory such as /usr/local/bin:

shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64
shell> cp bin/ndbd /usr/local/bin/ndbd
shell> cp bin/ndbmtd /usr/local/bin/ndbmtd

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from /var/tmp once ndb_mgm and ndb_mgmd have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

shell> cd /usr/local/bin
shell> chmod +x ndb*

The preceding steps should be repeated on each data node host.

Although only one of the data node executables is required to run an NDB Cluster data node, we have shown you how to install both ndbd and ndbmtd in the preceding instructions. We recommend that you do this when installing or upgrading NDB Cluster, even if you plan to use only one of them, since this saves time and trouble in the event that you later decide to change from one to the other.
Installing NDB Cluster from RPM

Management nodes. Installation of the management node does not require the mysqld binary. Only the NDB Cluster management server (ndb_mgmd) is required; you most likely want to install the management client (ndb_mgm) as well. Both of these binaries also be found in the .tar.gz archive. Again, we assume that you have placed this archive in /var/tmp.

As system root, perform the following steps to install ndb_mgm and ndb_mgmd on the management node host:

1. Change location to the /var/tmp directory, and extract the ndb_mgm and ndb_mgmd from the archive into a suitable directory such as /usr/local/bin:

```shell
shell> cd /var/tmp
shell> tar -zxvf mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64.tar.gz
shell> cd mysql-cluster-gpl-8.0.23-linux-glibc2.12-x86_64
shell> cp bin/ndb_mgm* /usr/local/bin
```

(You can safely delete the directory created by unpacking the downloaded archive, and the files it contains, from /var/tmp once ndb_mgm and ndb_mgmd have been copied to the executables directory.)

2. Change location to the directory into which you copied the files, and then make both of them executable:

```shell
shell> cd /usr/local/bin
shell> chmod +x ndb_mgm*
```

In Section 4.3, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

4.1.2 Installing NDB Cluster from RPM

This section covers the steps necessary to install the correct executables for each type of NDB Cluster 8.0 node using RPM packages supplied by Oracle. For information about RPMs for previous versions of NDB Cluster, see Installation using old-style RPMs (NDB 7.5.3 and earlier).

As an alternative to the method described in this section, Oracle provides MySQL Repositories for NDB Cluster that are compatible with many common Linux distributions. Two repositories, listed here, are available for RPM-based distributions:

- For distributions using yum or dnf, you can use the MySQL Yum Repository for NDB Cluster. See Installing MySQL NDB Cluster Using the Yum Repository, for instructions and additional information.

- For SLES, you can use the MySQL SLES Repository for NDB Cluster. See Installing MySQL NDB Cluster Using the SLES Repository, for instructions and additional information.

RPMs are available for both 32-bit and 64-bit Linux platforms. The filenames for these RPMs use the following pattern:

```
mysql-cluster-community-data-node-8.0.23-1.el7.x86_64.rpm
mysql-cluster-license-component-ver-rev.distro.arch.rpm
```

| license: | commercial | community |
|-----------------|-------------|
| component: | management-server | data-node | server | client | other-see text |
| ver: | major.minor.release |
| rev: | major.minor[.minor] |
| distro: | e16 | e17 | sles12 |
| arch: | i686 | x86_64 |
**license** indicates whether the RPM is part of a Commercial or Community release of NDB Cluster. In the remainder of this section, we assume for the examples that you are installing a Community release.

Possible values for **component**, with descriptions, can be found in the following table:

**Table 4.2 Components of the NDB Cluster RPM distribution**

<table>
<thead>
<tr>
<th>Component</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>auto-installer</td>
<td>NDB Cluster Auto Installer program; see Section 4.8, “The NDB Cluster Auto-Installer (No longer supported)” for usage</td>
</tr>
<tr>
<td>client</td>
<td>MySQL and NDB client programs; includes mysql client, ndb_mgm client, and other client tools</td>
</tr>
<tr>
<td>common</td>
<td>Character set and error message information needed by the MySQL server</td>
</tr>
<tr>
<td>data-node</td>
<td>ndbd and ndbmtd data node binaries</td>
</tr>
<tr>
<td>devel</td>
<td>Headers and library files needed for MySQL client development</td>
</tr>
<tr>
<td>embedded</td>
<td>Embedded MySQL server</td>
</tr>
<tr>
<td>embedded-compat</td>
<td>Backwards-compatible embedded MySQL server</td>
</tr>
<tr>
<td>embedded-devel</td>
<td>Header and library files for developing applications for embedded MySQL</td>
</tr>
<tr>
<td>java</td>
<td>JAR files needed for support of ClusterJ applications</td>
</tr>
<tr>
<td>libs</td>
<td>MySQL client libraries</td>
</tr>
<tr>
<td>libs-compat</td>
<td>Backwards-compatible MySQL client libraries</td>
</tr>
<tr>
<td>management-server</td>
<td>The NDB Cluster management server (ndb_mgmd)</td>
</tr>
<tr>
<td>memcached</td>
<td>Files needed to support ndbmemcache</td>
</tr>
<tr>
<td>minimal-debuginfo</td>
<td>Debug information for package server-minimal; useful when developing applications that use this package or when debugging this package</td>
</tr>
<tr>
<td>ndbclient</td>
<td>NDB client library for running NDB API and MGM API applications (libndbclient)</td>
</tr>
<tr>
<td>ndbclient-devel</td>
<td>Header and other files needed for developing NDB API and MGM API applications</td>
</tr>
<tr>
<td>nodejs</td>
<td>Files needed to set up Node.JS support for NDB Cluster</td>
</tr>
<tr>
<td>server</td>
<td>The MySQL server (mysqld) with NDB storage engine support included, and associated MySQL server programs</td>
</tr>
<tr>
<td>server-minimal</td>
<td>Minimal installation of the MySQL server for NDB and related tools</td>
</tr>
<tr>
<td>test</td>
<td>mysqltest, other MySQL test programs, and support files</td>
</tr>
</tbody>
</table>

A single bundle (.tar file) of all NDB Cluster RPMs for a given platform and architecture is also available. The name of this file follows the pattern shown here:

**mysql-cluster-license-ver-rev.distro.arch.rpm-bundle.tar**

You can extract the individual RPM files from this file using `tar` or your preferred tool for extracting archives.
Installing NDB Cluster from RPM

The components required to install the three major types of NDB Cluster nodes are given in the following list:

- **Management node**: management-server
- **Data node**: data-node
- **SQL node**: server and common

In addition, the **client** RPM should be installed to provide the ndb_mgm management client on at least one management node. You may also wish to install it on SQL nodes, to have mysql and other MySQL client programs available on these. We discuss installation of nodes by type later in this section.

`ver` represents the three-part NDB storage engine version number in 8.0.x format, shown as 8.0.23 in the examples. `rev` provides the RPM revision number in major.minor format. In the examples shown in this section, we use 1.1 for this value.

The `distro` (Linux distribution) is one of `rhel5` (Oracle Linux 5, Red Hat Enterprise Linux 4 and 5), `el6` (Oracle Linux 6, Red Hat Enterprise Linux 6), `el7` (Oracle Linux 7, Red Hat Enterprise Linux 7), or `sles12` (SUSE Enterprise Linux 12). For the examples in this section, we assume that the host runs Oracle Linux 7, Red Hat Enterprise Linux 7, or the equivalent (el7).

`arch` is `i686` for 32-bit RPMs and `x86_64` for 64-bit versions. In the examples shown here, we assume a 64-bit platform.

The NDB Cluster version number in the RPM file names (shown here as 8.0.23) can vary according to the version which you are actually using. **It is very important that all of the Cluster RPMs to be installed have the same version number.** The architecture should also be appropriate to the machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs (`x86_64`) cannot be used with 32-bit operating systems (use `i686` for the latter).

**Data nodes.** On a computer that is to host an NDB Cluster data node it is necessary to install only the `data-node` RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL website:

```sh
shell> rpm -Uhv mysql-cluster-community-data-node-8.0.23-1.el7.x86_64.rpm
```

This installs the ndbd and ndbmtd data node binaries in `/usr/sbin`. Either of these can be used to run a data node process on this host.

**SQL nodes.** Copy the `server` and `common` RPMs to each machine to be used for hosting an NDB Cluster SQL node (`server` requires `common`). Install the `server` RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL website:

```sh
shell> rpm -Uhv mysql-cluster-community-server-8.0.23-1.el7.x86_64.rpm
```

This installs the MySQL server binary (mysqld), with NDB storage engine support, in the `/usr/sbin` directory. It also installs all needed MySQL Server support files and useful MySQL server programs, including the `mysql.server` and `mysqld_safe` startup scripts (in `/usr/share/mysql` and `/usr/bin`, respectively). The RPM installer should take care of general configuration issues (such as creating the `mysql` user and group, if needed) automatically.

---

**Important**

You must use the versions of these RPMs released for NDB Cluster; those released for the standard MySQL server do not provide support for the NDB storage engine.
To administer the SQL node (MySQL server), you should also install the client RPM, as shown here:

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-8.0.23-1.sles11.i386.rpm
```

This installs the mysql client and other MySQL client programs, such as mysqladmin and mysqldump, to /usr/bin.

**Management nodes.** To install the NDB Cluster management server, it is necessary only to use the management-server RPM. Copy this RPM to the computer intended to host the management node, and then install it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the management-server RPM downloaded from the MySQL website):

```
shell> rpm -Uhv mysql-cluster-community-management-server-8.0.23-1.el7.x86_64.rpm
```

This RPM installs the management server binary ndb_mgmd in the /usr/sbin directory. While this is the only program actually required for running a management node, it is also a good idea to have the ndb_mgm NDB Cluster management client available as well. You can obtain this program, as well as other NDB client programs such as ndb_desc and ndb_config, by installing the client RPM as described previously.

See Installing MySQL on Linux Using RPM Packages from Oracle, for general information about installing MySQL using RPMs supplied by Oracle.

After installing from RPM, you still need to configure the cluster; see Section 4.3, “Initial Configuration of NDB Cluster”, for the relevant information.

*It is very important that all of the Cluster RPMs to be installed have the same version number.* The architecture designation should also be appropriate to the machine on which the RPM is to be installed; in particular, you should keep in mind that 64-bit RPMs cannot be used with 32-bit operating systems.

**Data nodes.** On a computer that is to host a cluster data node it is necessary to install only the server RPM. To do so, copy this RPM to the data node host, and run the following command as the system root user, replacing the name shown for the RPM as necessary to match that of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.23-1.sles11.i386.rpm
```

Although this installs all NDB Cluster binaries, only the program ndbd or ndbmdt (both in /usr/sbin) is actually needed to run an NDB Cluster data node.

**SQL nodes.** On each machine to be used for hosting a cluster SQL node, install the server RPM by executing the following command as the system root user, replacing the name shown for the RPM as necessary to match the name of the RPM downloaded from the MySQL website:

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.23-1.sles11.i386.rpm
```

This installs the MySQL server binary (mysqld) with NDB storage engine support in the /usr/sbin directory, as well as all needed MySQL Server support files. It also installs the mysql.server and mysqld_safe startup scripts (in /usr/share/mysql and /usr/bin, respectively). The RPM installer should take care of general configuration issues (such as creating the mysql user and group, if needed) automatically.

To administer the SQL node (MySQL server), you should also install the client RPM, as shown here:

```
shell> rpm -Uhv MySQL-Cluster-client-gpl-8.0.23-1.sles11.i386.rpm
```

This installs the mysql client program.

**Management nodes.** To install the NDB Cluster management server, it is necessary only to use the server RPM. Copy this RPM to the computer intended to host the management node, and then install
Installing NDB Cluster Using .deb Files

it by running the following command as the system root user (replace the name shown for the RPM as necessary to match that of the server RPM downloaded from the MySQL website):

```
shell> rpm -Uhv MySQL-Cluster-server-gpl-8.0.23-1.sles11.i386.rpm
```

Although this RPM installs many other files, only the management server binary `ndb_mgmd` (in the `/usr/sbin` directory) is actually required for running a management node. The server RPM also installs `ndb_mgm`, the NDB management client.


4.1.3 Installing NDB Cluster Using .deb Files

The section provides information about installing NDB Cluster on Debian and related Linux distributions such Ubuntu using the .deb files supplied by Oracle for this purpose.

Oracle also provides an NDB Cluster APT repository for Debian and other distributions. See Installing MySQL NDB Cluster Using the APT Repository, for instructions and additional information.

Oracle provides .deb installer files for NDB Cluster for 32-bit and 64-bit platforms. For a Debian-based system, only a single installer file is necessary. This file is named using the pattern shown here, according to the applicable NDB Cluster version, Debian version, and architecture:

```
mysql-cluster-gpl-ndbver-debiandebianver-arch.deb
```

Here, `ndbver` is the 3-part NDB engine version number, `debianver` is the major version of Debian (8 or 9), and `arch` is one of `i686` or `x86_64`. In the examples that follow, we assume you wish to install NDB 8.0.23 on a 64-bit Debian 9 system; in this case, the installer file is named `mysql-cluster-gpl-8.0.23-debian9-x86_64.deb-bundle.tar`.

Once you have downloaded the appropriate .deb file, you can untar it, and then install it from the command line using `dpkg`, like this:

```
shell> dpkg -i mysql-cluster-gpl-8.0.23-debian9-i686.deb
```

You can also remove it using `dpkg` as shown here:

```
shell> dpkg -r mysql
```

The installer file should also be compatible with most graphical package managers that work with .deb files, such as GDebi for the Gnome desktop.

The .deb file installs NDB Cluster under `/opt/mysql/server-version/`, where `version` is the 2-part release series version for the included MySQL server. For NDB 8.0, this is always 5.7. The directory layout is the same as that for the generic Linux binary distribution (see MySQL Installation Layout for Generic Unix/Linux Binary Package), with the exception that startup scripts and configuration files are found in `support-files` instead of `share`. All NDB Cluster executables, such as `ndb_mgm`, `ndbd`, and `ndb_mgmd`, are placed in the `bin` directory.

4.1.4 Building NDB Cluster from Source on Linux

This section provides information about compiling NDB Cluster on Linux and other Unix-like platforms. Building NDB Cluster from source is similar to building the standard MySQL Server, although it differs in a few key respects discussed here. For general information about building MySQL from source, see Installing MySQL from Source. For information about compiling NDB Cluster on Windows platforms, see Section 4.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

Building MySQL NDB Cluster 8.0 requires using the MySQL Server 8.0 sources. These are available from the MySQL downloads page at https://dev.mysql.com/downloads/. The archived source file should
have a name similar to mysql-8.0.23.tar.gz. You can also obtain the sources from GitHub at https://github.com/mysql/mysql-server.

### Note

In previous versions, building of NDB Cluster from standard MySQL Server sources was not supported. In MySQL 8.0 and NDB Cluster 8.0, this is no longer the case—both products are now built from the same sources.

The `WITH_NDBCLUSTER` option for `CMake` causes the binaries for the management nodes, data nodes, and other NDB Cluster programs to be built; it also causes `mysqld` to be compiled with NDB storage engine support. This option (or one of its aliases `WITH_NDBCLUSTER_STORAGE_ENGINE` and `WITH_PLUGIN_NDBCLUSTER`) is required when building NDB Cluster.

### Important

The `WITH_NDB_JAVA` option is enabled by default. This means that, by default, if `CMake` cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using `WITH_NDB_JAVA=OFF`. Use `WITH_CLASSPATH` to provide the Java classpath if needed.

For more information about `CMake` options specific to building NDB Cluster, see [Options for Compiling NDB Cluster](#).

After you have run `make && make install` (or your system’s equivalent), the result is similar to what is obtained by unpacking a precompiled binary to the same location.

#### Management nodes.

When building from source and running the default `make install`, the management server and management client binaries (`ndb_mgmd` and `ndb_mgm`) can be found in `/usr/local/mysql/bin`. Only `ndb_mgmd` is required to be present on a management node host; however, it is also a good idea to have `ndb_mgm` present on the same host machine. Neither of these executables requires a specific location on the host machine’s file system.

#### Data nodes.

The only executable required on a data node host is the data node binary `ndbd` or `ndbmt`. (`mysqld`, for example, does not have to be present on the host machine.) By default, when building from source, this file is placed in the directory `/usr/local/mysql/bin`. For installing on multiple data node hosts, only `ndbd` or `ndbmt` need be copied to the other host machine or machines. (This assumes that all data node hosts use the same architecture and operating system; otherwise you may need to compile separately for each different platform.) The data node binary need not be in any particular location on the host’s file system, as long as the location is known.

When compiling NDB Cluster from source, no special options are required for building multithreaded data node binaries. Configuring the build with NDB storage engine support causes `ndbmt` to be built automatically; `make install` places the `ndbmt` binary in the installation bin directory along with `mysqld`, `ndbd`, and `ndb_mgm`.

#### SQL nodes.

If you compile MySQL with clustering support, and perform the default installation (using `make install` as the system root user), `mysqld` is placed in `/usr/local/mysql/bin`. Follow the steps given in [Installing MySQL from Source](#) to make `mysqld` ready for use. If you want to run multiple SQL nodes, you can use a copy of the same `mysqld` executable and its associated support files on several machines. The easiest way to do this is to copy the entire `/usr/local/mysql` directory and all directories and files contained within it to the other SQL node host or hosts, then repeat the steps from [Installing MySQL from Source](#) on each machine. If you configure the build with a nondefault `PREFIX` option, you must adjust the directory accordingly.

In Section 4.3, “Initial Configuration of NDB Cluster”, we create configuration files for all of the nodes in our example NDB Cluster.

### 4.2 Installing NDB Cluster on Windows
This section describes installation procedures for NDB Cluster on Windows hosts. NDB Cluster 8.0 binaries for Windows can be obtained from https://dev.mysql.com/downloads/cluster/. For information about installing NDB Cluster on Windows from a binary release provided by Oracle, see Section 4.2.1, “Installing NDB Cluster on Windows from a Binary Release”.

It is also possible to compile and install NDB Cluster from source on Windows using Microsoft Visual Studio. For more information, see Section 4.2.2, “Compiling and Installing NDB Cluster from Source on Windows”.

4.2.1 Installing NDB Cluster on Windows from a Binary Release

This section describes a basic installation of NDB Cluster on Windows using a binary “no-install” NDB Cluster release provided by Oracle, using the same 4-node setup outlined in the beginning of this section (see Chapter 4, NDB Cluster Installation), as shown in the following table:

<table>
<thead>
<tr>
<th>Node</th>
<th>IP Address</th>
</tr>
</thead>
<tbody>
<tr>
<td>Management node (mgmd)</td>
<td>198.51.100.10</td>
</tr>
<tr>
<td>SQL node (mysqld)</td>
<td>198.51.100.20</td>
</tr>
<tr>
<td>Data node &quot;A&quot; (ndbd)</td>
<td>198.51.100.30</td>
</tr>
<tr>
<td>Data node &quot;B&quot; (ndbd)</td>
<td>198.51.100.40</td>
</tr>
</tbody>
</table>

As on other platforms, the NDB Cluster host computer running an SQL node must have installed on it a MySQL Server binary (mysqld.exe). You should also have the MySQL client (mysql.exe) on this host. For management nodes and data nodes, it is not necessary to install the MySQL Server binary; however, each management node requires the management server daemon (ndb_mgmd.exe); each data node requires the data node daemon (ndbd.exe or ndbmd.exe). For this example, we refer to ndbd.exe as the data node executable, but you can install ndbmd.exe, the multithreaded version of this program, instead, in exactly the same way. You should also install the management client (ndb_mgm.exe) on the management server host. This section covers the steps necessary to install the correct Windows binaries for each type of NDB Cluster node.

**Note**

As with other Windows programs, NDB Cluster executables are named with the .exe file extension. However, it is not necessary to include the .exe extension when invoking these programs from the command line. Therefore, we often simply refer to these programs in this documentation as mysqld, mysql, ndb_mgmd, and so on. You should understand that, whether we refer (for example) to mysqld or mysqld.exe, either name means the same thing (the MySQL Server program).

For setting up an NDB Cluster using Oracles’s no-install binaries, the first step in the installation process is to download the latest NDB Cluster Windows ZIP binary archive from https://dev.mysql.com/downloads/cluster/. This archive has a filename of the mysql-cluster-gpl-ver-winarch.zip, where ver is the NDB storage engine version (such as 8.0.23), and arch is the architecture (32 for 32-bit binaries, and 64 for 64-bit binaries). For example, the NDB Cluster 8.0.23 archive for 64-bit Windows systems is named mysql-cluster-gpl-8.0.23-win64.zip.

You can run 32-bit NDB Cluster binaries on both 32-bit and 64-bit versions of Windows; however, 64-bit NDB Cluster binaries can be used only on 64-bit versions of Windows. If you are using a 32-bit version of Windows on a computer that has a 64-bit CPU, then you must use the 32-bit NDB Cluster binaries.

To minimize the number of files that need to be downloaded from the Internet or copied between machines, we start with the computer where you intend to run the SQL node.
Installing NDB Cluster on Windows from a Binary Release

**SQL node.** We assume that you have placed a copy of the archive in the directory `C:\Documents and Settings\username\My Documents\Downloads` on the computer having the IP address 198.51.100.20, where `username` is the name of the current user. (You can obtain this name using `ECHO %USERNAME%` on the command line.) To install and run NDB Cluster executables as Windows services, this user should be a member of the `Administrators` group.

Extract all the files from the archive. The Extraction Wizard integrated with Windows Explorer is adequate for this task. (If you use a different archive program, be sure that it extracts all files and directories from the archive, and that it preserves the archive's directory structure.) When you are asked for a destination directory, enter `C:\`, which causes the Extraction Wizard to extract the archive to the directory `C:\mysql-cluster-gpl-ver-winarch`. Rename this directory to `C:\mysql`.

It is possible to install the NDB Cluster binaries to directories other than `C:\mysql\bin`; however, if you do so, you must modify the paths shown in this procedure accordingly. In particular, if the MySQL Server (SQL node) binary is installed to a location other than `C:\mysql` or `C:\Program Files\MySQL\MySQL Server 8.0`, or if the SQL node's data directory is in a location other than `C:\mysql\data` or `C:\Program Files\MySQL\MySQL Server 8.0\data`, extra configuration options must be used on the command line or added to the `my.ini` or `my.cnf` file when starting the SQL node. For more information about configuring a MySQL Server to run in a nonstandard location, see Installing MySQL on Microsoft Windows Using a noinstall ZIP Archive.

For a MySQL Server with NDB Cluster support to run as part of an NDB Cluster, it must be started with the options `--ndbcluster` and `--ndb-connectstring`. While you can specify these options on the command line, it is usually more convenient to place them in an option file. To do this, create a new text file in Notepad or another text editor. Enter the following configuration information into this file:

```
[mysqld]
# Options for mysqld process:
ndbcluster                       # run NDB storage engine
ndb-connectstring=198.51.100.10  # location of management server
```

You can add other options used by this MySQL Server if desired (see Creating an Option File), but the file must contain the options shown, at a minimum. Save this file as `C:\mysql\my.ini`. This completes the installation and setup for the SQL node.

**Data nodes.** An NDB Cluster data node on a Windows host requires only a single executable, one of either `ndbd.exe` or `ndbmtd.exe`. For this example, we assume that you are using `ndbd.exe`, but the same instructions apply when using `ndbmtd.exe`. On each computer where you wish to run a data node (the computers having the IP addresses 198.51.100.30 and 198.51.100.40), create the directories `C:\mysql`, `C:\mysql\bin`, and `C:\mysql\cluster-data`; then, on the computer where you downloaded and extracted the no-install archive, locate `ndbd.exe` in the `C:\mysql\bin` directory. Copy this file to the `C:\mysql\bin` directory on each of the two data node hosts.

To function as part of an NDB Cluster, each data node must be given the address or hostname of the management server. You can supply this information on the command line using the `--ndb-connectstring` or `-c` option when starting each data node process. However, it is usually preferable to put this information in an option file. To do this, create a new text file in Notepad or another text editor and enter the following text:

```
[mysql_cluster]
# Options for data node process:
ndb-connectstring=198.51.100.10  # location of management server
```

Save this file as `C:\mysql\my.ini` on the data node host. Create another text file containing the same information and save it on as `C:\mysql\my.ini` on the other data node host, or copy the my.ini file from the first data node host to the second one, making sure to place the copy in the second data node's `C:\mysql` directory. Both data node hosts are now ready to be used in the NDB Cluster, which leaves only the management node to be installed and configured.

**Management node.** The only executable program required on a computer used for hosting an NDB Cluster management node is the management server program `ndb_mgmd.exe`. However, in
order to administer the NDB Cluster once it has been started, you should also install the NDB Cluster
management client program `ndb_mgm.exe` on the same machine as the management server. Locate
these two programs on the machine where you downloaded and extracted the no-install archive;
this should be the directory `C:\mysql\bin` on the SQL node host. Create the directory `C:\mysql\bin`
on the computer having the IP address 198.51.100.10, then copy both programs to this directory.

You should now create two configuration files for use by `ndb_mgm.exe`:

1. A local configuration file to supply configuration data specific to the management node itself.
   Typically, this file needs only to supply the location of the NDB Cluster global configuration file (see
   item 2).

   To create this file, start a new text file in Notepad or another text editor, and enter the following
   information:

   ```
   [mysql_cluster]
   # Options for management node process
   config-file=C:/mysql/bin/config.ini
   ```

   Save this file as the text file `C:\mysql\bin\my.ini`.

2. A global configuration file from which the management node can obtain configuration information
governing the NDB Cluster as a whole. At a minimum, this file must contain a section for each node
in the NDB Cluster, and the IP addresses or hostnames for the management node and all data
nodes (`HostName` configuration parameter). It is also advisable to include the following additional
information:

   - The IP address or hostname of any SQL nodes
   - The data memory and index memory allocated to each data node (`DataMemory` and
     `IndexMemory` configuration parameters)
   - The number of fragment replicas, using the `NoOfReplicas` configuration parameter (see
     Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”)
   - The directory where each data node stores its data and log file, and the directory where the
     management node keeps its log files (in both cases, the `DataDir` configuration parameter)

   Create a new text file using a text editor such as Notepad, and input the following information:

   ```
   [ndbd default]
   # Options affecting ndbd processes on all data nodes:
   NoOfReplicas=2                      # Number of fragment replicas
   DataDir=C:/mysql/cluster-data       # Directory for each data node's data files
   DataMemory=80M    # Memory allocated to data storage
   IndexMemory=18M   # Memory allocated to index storage
   # For DataMemory and IndexMemory, we have used the
   # default values. Since the "world" database takes up
   # only about 500KB, this should be more than enough for
   # this example Cluster setup.
   [ndb_mgmd]
   # Management process options:
   DataDir=C:/mysql/bin/cluster-logs   # Directory for management node log files
   [ndbd]
   # Options for data node "A":
   # (one [ndbd] section per data node)
   HostName=198.51.100.30          # Hostname or IP address
   [ndbd]
   # Options for data node "A":
   HostName=198.51.100.40           # Hostname or IP address
   # SQL node options:
   ```
Compiling and Installing NDB Cluster from Source on Windows

Important

A single backslash character (\) cannot be used when specifying directory paths in program options or configuration files used by NDB Cluster on Windows. Instead, you must either escape each backslash character with a second backslash (\\), or replace the backslash with a forward slash character (/). For example, the following line from the [ndb_mgmd] section of an NDB Cluster config.ini file does not work:

```
DataDir=C:\mysql\bin\cluster-logs
```

Instead, you may use either of the following:

```
DataDir=C:\\mysql\\bin\\cluster-logs  # Escaped backslashes
```

```
DataDir=C:/mysql/bin/cluster-logs     # Forward slashes
```

For reasons of brevity and legibility, we recommend that you use forward slashes in directory paths used in NDB Cluster program options and configuration files on Windows.

4.2.2 Compiling and Installing NDB Cluster from Source on Windows

Oracle provides precompiled NDB Cluster binaries for Windows which should be adequate for most users. However, if you wish, it is also possible to compile NDB Cluster for Windows from source code. The procedure for doing this is almost identical to the procedure used to compile the standard MySQL Server binaries for Windows, and uses the same tools. However, there are two major differences:

• Building MySQL NDB Cluster 8.0 requires using the MySQL Server 8.0 sources. These are available from the MySQL downloads page at https://dev.mysql.com/downloads/. The archived source file should have a name similar to mysql-8.0.23.tar.gz. You can also obtain the sources from GitHub at https://github.com/mysql/mysql-server.

• You must configure the build using the WITH_NDBCLUSTER option in addition to any other build options you wish to use with CMake. WITH_NDBCLUSTER_STORAGE_ENGINE and WITH_PLUGIN_NDBCLUSTER are supported as aliases for WITH_NDBCLUSTER, and work in exactly the same way.

Important

The WITH_NDB_JAVA option is enabled by default. This means that, by default, if CMake cannot find the location of Java on your system, the configuration process fails; if you do not wish to enable Java and ClusterJ support, you must indicate this explicitly by configuring the build using 

```
-DWITH_NDB_JAVA=OFF
```

(Bug #12379735) Use WITH_CLASSPATH to provide the Java classpath if needed.

For more information about CMake options specific to building NDB Cluster, see Options for Compiling NDB Cluster.

Once the build process is complete, you can create a Zip archive containing the compiled binaries; Installing MySQL Using a Standard Source Distribution provides the commands needed to perform this task on Windows systems. The NDB Cluster binaries can be found in the bin directory of the resulting archive, which is equivalent to the no-install archive, and which can be installed and configured in the same manner. For more information, see Section 4.2.1, “Installing NDB Cluster on Windows from a Binary Release”.

4.2.3 Initial Startup of NDB Cluster on Windows
Once the NDB Cluster executables and needed configuration files are in place, performing an initial start of the cluster is simply a matter of starting the NDB Cluster executables for all nodes in the cluster. Each cluster node process must be started separately, and on the host computer where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes.

1. On the management node host, issue the following command from the command line to start the management node process. The output should appear similar to what is shown here:

```plaintext
C:\mysql\bin> ndb_mgmd
2010-06-23 07:53:34 [MgmtSrvr] INFO -- Reading cluster configuration from 'config.ini'
```

The management node process continues to print logging output to the console. This is normal, because the management node is not running as a Windows service. (If you have used NDB Cluster on a Unix-like platform such as Linux, you may notice that the management node's default behavior in this regard on Windows is effectively the opposite of its behavior on Unix systems, where it runs by default as a Unix daemon process. This behavior is also true of NDB Cluster data node processes running on Windows.) For this reason, do not close the window in which ndb_mgmd.exe is running; doing so kills the management node process. (See Section 4.2.4, “Installing NDB Cluster Processes as Windows Services”, where we show how to install and run NDB Cluster processes as Windows services.)

The required `-f` option tells the management node where to find the global configuration file (config.ini). The long form of this option is `--config-file`.

Important

An NDB Cluster management node caches the configuration data that it reads from config.ini; once it has created a configuration cache, it ignores the config.ini file on subsequent starts unless forced to do otherwise. This means that, if the management node fails to start due to an error in this file, you must make the management node re-read config.ini after you have corrected any errors in it. You can do this by starting ndb_mgmd.exe with the `--reload` or `--initial` option on the command line. Either of these options works to refresh the configuration cache.

It is not necessary or advisable to use either of these options in the management node's my.ini file.

For additional information about options which can be used with ndb_mgmd, see Section 6.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”, as well as Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

2. On each of the data node hosts, run the command shown here to start the data node processes:

```plaintext
C:\mysql\bin> ndbd
```

In each case, the first line of output from the data node process should resemble what is shown in the preceding example, and is followed by additional lines of logging output. As with the management node process, this is normal, because the data node is not running as a Windows service. For this reason, do not close the console window in which the data node process is running; doing so kills ndbd.exe. (For more information, see Section 4.2.4, “Installing NDB Cluster Processes as Windows Services”)

3. Do not start the SQL node yet; it cannot connect to the cluster until the data nodes have finished starting, which may take some time. Instead, in a new console window on the management node host, start the NDB Cluster management client ndb_mgm.exe, which should be in C:\mysql\bin on the management node host. (Do not try to re-use the console window where ndb_mgmd.exe is
running by typing CTRL+C, as this kills the management node.) The resulting output should look like this:

```
C:\mysql\bin> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm>
```

When the prompt `ndb_mgm>` appears, this indicates that the management client is ready to receive NDB Cluster management commands. You can observe the status of the data nodes as they start by entering `ALL STATUS` at the management client prompt. This command causes a running report of the data nodes's startup sequence, which should look something like this:

```
ndb_mgm> ALL STATUS
Connected to Management Server at: localhost:1186
Node 2: starting (Last completed phase 3) (mysql-8.0.24-ndb-8.0.24)
Node 3: starting (Last completed phase 3) (mysql-8.0.24-ndb-8.0.24)
Node 2: starting (Last completed phase 4) (mysql-8.0.24-ndb-8.0.24)
Node 3: starting (Last completed phase 4) (mysql-8.0.24-ndb-8.0.24)
Node 2: Started (version 8.0.24)
Node 3: Started (version 8.0.24)
ndb_mgm>
```

**Note**

Commands issued in the management client are not case-sensitive; we use uppercase as the canonical form of these commands, but you are not required to observe this convention when inputting them into the `ndb_mgm` client. For more information, see Section 7.1, “Commands in the NDB Cluster Management Client”.

The output produced by `ALL STATUS` is likely to vary from what is shown here, according to the speed at which the data nodes are able to start, the release version number of the NDB Cluster software you are using, and other factors. What is significant is that, when you see that both data nodes have started, you are ready to start the SQL node.

You can leave `ndb_mgm.exe` running; it has no negative impact on the performance of the NDB Cluster, and we use it in the next step to verify that the SQL node is connected to the cluster after you have started it.

4. On the computer designated as the SQL node host, open a console window and navigate to the directory where you unpacked the NDB Cluster binaries (if you are following our example, this is `C:\mysql\bin`).

Start the SQL node by invoking `mysqld.exe` from the command line, as shown here:

```
C:\mysql\bin> mysqld --console
```

The `--console` option causes logging information to be written to the console, which can be helpful in the event of problems. (Once you are satisfied that the SQL node is running in a satisfactory manner, you can stop it and restart it without the `--console` option, so that logging is performed normally.)

In the console window where the management client (`ndb_mgm.exe`) is running on the management node host, enter the `SHOW` command, which should produce output similar to what is shown here:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
| ndbd(NDDB) | 2 node(s) |
| id=2       | 198.51.100.30 (Version: 8.0.24-ndb-8.0.24, Nodegroup: 0, *) |
| id=3       | 198.51.100.40 (Version: 8.0.24-ndb-8.0.24, Nodegroup: 0) |
```

71
You can also verify that the SQL node is connected to the NDB Cluster in the **mysql** client (**mysql.exe**) using the **SHOW ENGINE NDB STATUS** statement.

You should now be ready to work with database objects and data using NDB Cluster’s **NDBCLUSTER** storage engine. See Section 4.5, “NDB Cluster Example with Tables and Data”, for more information and examples.

You can also install **ndb_mgmd.exe**, **ndbd.exe**, and **ndbmtd.exe** as Windows services. For information on how to do this, see Section 4.2.4, “Installing NDB Cluster Processes as Windows Services”.

### 4.2.4 Installing NDB Cluster Processes as Windows Services

Once you are satisfied that NDB Cluster is running as desired, you can install the management nodes and data nodes as Windows services, so that these processes are started and stopped automatically whenever Windows is started or stopped. This also makes it possible to control these processes from the command line with the appropriate **SC START** and **SC STOP** commands, or using the Windows graphical **Services** utility. **NET START** and **NET STOP** commands can also be used.

Installing programs as Windows services usually must be done using an account that has Administrator rights on the system.

To install the management node as a service on Windows, invoke **ndb_mgmd.exe** from the command line on the machine hosting the management node, using the **--install** option, as shown here:

```bash
C:\> C:\mysql\bin\ndb_mgmd.exe --install
Installing service 'NDB Cluster Management Server'
as '"C:\mysql\bin\ndbd.exe"'"--service=ndb_mgmd"'
Service successfully installed.
```

**Important**

When installing an NDB Cluster program as a Windows service, you should always specify the complete path; otherwise the service installation may fail with the error **The system cannot find the file specified**.

The **--install** option must be used first, ahead of any other options that might be specified for **ndb_mgmd.exe**. However, it is preferable to specify such options in an options file instead. If your options file is not in one of the default locations as shown in the output of **ndb_mgmd.exe --help**, you can specify the location using the **--config-file** option.

Now you should be able to start and stop the management server like this:

```bash
C:\> SC START ndb_mgmd
C:\> SC STOP ndb_mgmd
```

**Note**

If using **NET** commands, you can also start or stop the management server as a Windows service using the descriptive name, as shown here:

```bash
C:\> NET START 'NDB Cluster Management Server'
The NDB Cluster Management Server service is starting.
The NDB Cluster Management Server service was started successfully.
C:\> NET STOP 'NDB Cluster Management Server'
The NDB Cluster Management Server service is stopping..
The NDB Cluster Management Server service was stopped successfully.
```

It is usually simpler to specify a short service name or to permit the default service name to be used when installing the service, and then reference that name when starting or stopping the service. To
specify a service name other than `ndb_mgmd`, append it to the `--install` option, as shown in this example:

```bash
C:\> C:\mysql\bin\ndb_mgmd.exe --install=mgmd1
Installing service 'NDB Cluster Management Server' as "C:\mysql\bin\ndb_mgmd.exe" "--service=mgmd1"
Service successfully installed.
```

Now you should be able to start or stop the service using the name you have specified, like this:

```bash
C:\> SC START mgmd1
C:\> SC STOP mgmd1
```

To remove the management node service, use `SC DELETE service_name`:

```bash
C:\> SC DELETE mgmd1
```

Alternatively, invoke `ndb_mgmd.exe` with the `--remove` option, as shown here:

```bash
C:\> C:\mysql\bin\ndb_mgmd.exe --remove
Removing service 'NDB Cluster Management Server'
Service successfully removed.
```

If you installed the service using a service name other than the default, pass the service name as the value of the `ndb_mgmd.exe --remove` option, like this:

```bash
C:\> C:\mysql\bin\ndb_mgmd.exe --remove=mgmd1
Removing service 'mgmd1'
Service successfully removed.
```

Installation of an NDB Cluster data node process as a Windows service can be done in a similar fashion, using the `--install` option for `ndbd.exe` (or `ndbmtd.exe`), as shown here:

```bash
C:\> C:\mysql\bin\ndbd.exe --install
Installing service 'NDB Cluster Data Node Daemon' as "C:\mysql\bin\ndbd.exe" "--service=ndbd"
Service successfully installed.
```

Now you can start or stop the data node as shown in the following example:

```bash
C:\> SC START ndbd
C:\> SC STOP ndbd
```

To remove the data node service, use `SC DELETE service_name`:

```bash
C:\> SC DELETE ndbd
```

Alternatively, invoke `ndbd.exe` with the `--remove` option, as shown here:

```bash
C:\> C:\mysql\bin\ndbd.exe --remove
Removing service 'NDB Cluster Data Node Daemon'
Service successfully removed.
```

As with `ndb_mgmd.exe` (and `mysqld.exe`), when installing `ndbd.exe` as a Windows service, you can also specify a name for the service as the value of the `--install`, and then use it when starting or stopping the service, like this:

```bash
C:\> C:\mysql\bin\ndbd.exe --install=dnode1
Installing service 'dnode1' as "C:\mysql\bin\ndbd.exe" "--service=dnode1"
Service successfully installed.
```

```bash
C:\> SC START dnode1
C:\> SC STOP dnode1
```

If you specified a service name when installing the data node service, you can use this name when removing it as well, as shown here:

```bash
C:\> SC DELETE dnode1
```

Alternatively, you can pass the service name as the value of the `ndbd.exe --remove` option, as shown here:
Initial Configuration of NDB Cluster

4.3 Initial Configuration of NDB Cluster

In this section, we discuss manual configuration of an installed NDB Cluster by creating and editing configuration files.

NDB Cluster also provides a GUI installer which can be used to perform the configuration without the need to edit text files in a separate application. For more information, see Section 4.8, “The NDB Cluster Auto-Installer (No longer supported)”.

For our four-node, four-host NDB Cluster (see Cluster nodes and host computers), it is necessary to write four configuration files, one per node host.

• Each data node or SQL node requires a my.cnf file that provides two pieces of information: a connection string that tells the node where to find the management node, and a line telling the MySQL server on this host (the machine hosting the data node) to enable the NDBCLUSTER storage engine.

For more information on connection strings, see Section 5.3.3, “NDB Cluster Connection Strings”.

• The management node needs a config.ini file telling it how many fragment replicas to maintain, how much memory to allocate for data and indexes on each data node, where to find the data nodes, where to save data to disk on each data node, and where to find any SQL nodes.

Configuring the data nodes and SQL nodes. The my.cnf file needed for the data nodes is fairly simple. The configuration file should be located in the /etc directory and can be edited using any text editor. (Create the file if it does not exist.) For example:

```
shell> vi /etc/my.cnf
```

Note

We show vi being used here to create the file, but any text editor should work just as well.

For each data node and SQL node in our example setup, my.cnf should look like this:

```
[mysqld]
# Options for mysqld process:
ndbcluster                      # run NDB storage engine

[mysql_cluster]
# Options for NDB Cluster processes:
ndb-connectstring=198.51.100.10  # location of management server
```

After entering the preceding information, save this file and exit the text editor. Do this for the machines hosting data node “A”, data node “B”, and the SQL node.

Important

Once you have started a mysqld process with the ndbcluster and ndb-connectstring parameters in the [mysqld] and [mysql_cluster] sections of the my.cnf file as shown previously, you cannot execute any CREATE TABLE or ALTER TABLE statements without having actually started the cluster. Otherwise, these statements fail with an error. This is by design.
Configuring the management node. The first step in configuring the management node is to create the directory in which the configuration file can be found and then to create the file itself. For example (running as `root`):

```
shell> mkdir /var/lib/mysql-cluster
shell> cd /var/lib/mysql-cluster
shell> vi config.ini
```

For our representative setup, the `config.ini` file should read as follows:

```
[nxdbd default]
# Options affecting ndbd processes on all data nodes:
NoOfReplicas=2   # Number of fragment replicas
DataMemory=98M    # How much memory to allocate for data storage

[ndb_mgmd]
# Management process options:
HostName=198.51.100.10          # Hostname or IP address of management node
DataDir=/var/lib/mysql-cluster  # Directory for management node log files

[ndbd]
# Options for data node "A":
HostName=198.51.100.30          # Hostname or IP address
NodeId=2                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node’s data files

[ndbd]
# Options for data node "B":
HostName=198.51.100.40          # Hostname or IP address
NodeId=3                        # Node ID for this data node
DataDir=/usr/local/mysql/data   # Directory for this data node’s data files

[mysqld]
# SQL node options:
HostName=198.51.100.20          # Hostname or IP address
# (additional mysqld connections can be
# specified for this node for various
# purposes such as running ndb_restore)
```

**Note**

The `world` database can be downloaded from [https://dev.mysql.com/doc/index-other.html](https://dev.mysql.com/doc/index-other.html).

After all the configuration files have been created and these minimal options have been specified, you are ready to proceed with starting the cluster and verifying that all processes are running. We discuss how this is done in Section 4.4, “Initial Startup of NDB Cluster”.

For more detailed information about the available NDB Cluster configuration parameters and their uses, see Section 5.3, “NDB Cluster Configuration Files”, and Chapter 5, *Configuration of NDB Cluster*. For configuration of NDB Cluster as relates to making backups, see Section 7.8.3, “Configuration for NDB Cluster Backups”.

**Note**

The default port for Cluster management nodes is 1186; the default port for data nodes is 2202. However, the cluster can automatically allocate ports for data nodes from those that are already free.

### 4.4 Initial Startup of NDB Cluster

Starting the cluster is not very difficult after it has been configured. Each cluster node process must be started separately, and on the host where it resides. The management node should be started first, followed by the data nodes, and then finally by any SQL nodes:

1. On the management host, issue the following command from the system shell to start the management node process:

```
shell> ndb_mgmd -f /var/lib/mysql-cluster/config.ini
```
The first time that it is started, \texttt{ndb_mgmd} must be told where to find its configuration file, using the \	exttt{-f} or \texttt{--config-file} option. (See Section 6.4, “\texttt{ndb_mgmd} — The NDB Cluster Management Server Daemon”, for details.)

For additional options which can be used with \texttt{ndb_mgmd}, see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

2. On each of the data node hosts, run this command to start the \texttt{ndbd} process:

\begin{verbatim}
shell> ndbd
\end{verbatim}

3. If you used RPM files to install MySQL on the cluster host where the SQL node is to reside, you can (and should) use the supplied startup script to start the MySQL server process on the SQL node.

If all has gone well, and the cluster has been set up correctly, the cluster should now be operational. You can test this by invoking the \texttt{ndb_mgm} management node client. The output should look like that shown here, although you might see some slight differences in the output depending upon the exact version of MySQL that you are using:

\begin{verbatim}
shell> ndb_mgm
-- NDB Cluster -- Management Client --
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
---------------------
[ndbd(NDB)]     2 node(s)
id=2    @198.51.100.30  (Version: 8.0.24-ndb-8.0.24, Nodegroup: 0, *)
id=3    @198.51.100.40  (Version: 8.0.24-ndb-8.0.24, Nodegroup: 0)
[ndb_mgmd(MGM)] 1 node(s)
id=1    @198.51.100.10  (Version: 8.0.24-ndb-8.0.24)
[mysqld(API)]   1 node(s)
id=4    @198.51.100.20  (Version: 8.0.24-ndb-8.0.24)
\end{verbatim}

The SQL node is referenced here as \texttt{[mysqld(API)]}, which reflects the fact that the \texttt{mysqld} process is acting as an NDB Cluster API node.

\begin{note}
The IP address shown for a given NDB Cluster SQL or other API node in the output of \texttt{SHOW} is the address used by the SQL or API node to connect to the cluster data nodes, and not to any management node.
\end{note}

You should now be ready to work with databases, tables, and data in NDB Cluster. See Section 4.5, “NDB Cluster Example with Tables and Data”, for a brief discussion.

4.5 NDB Cluster Example with Tables and Data

\begin{note}
The information in this section applies to NDB Cluster running on both Unix and Windows platforms.
\end{note}

Working with database tables and data in NDB Cluster is not much different from doing so in standard MySQL. There are two key points to keep in mind:

• For a table to be replicated in the cluster, it must use the \texttt{NDBCLUSTER} storage engine. To specify this, use the \texttt{ENGINE=NDBCLUSTER} or \texttt{ENGINE=NDB} option when creating the table:

\begin{verbatim}
CREATE TABLE tbl_name (col_name column_definitions) ENGINE=NDBCLUSTER;
\end{verbatim}

Alternatively, for an existing table that uses a different storage engine, use \texttt{ALTER TABLE} to change the table to use \texttt{NDBCLUSTER}:
ALTER TABLE `tbl_name` ENGINE=NDBCLUSTER;

- Every NDBCLUSTER table has a primary key. If no primary key is defined by the user when a table is created, the NDBCLUSTER storage engine automatically generates a hidden one. Such a key takes up space just as does any other table index. (It is not uncommon to encounter problems due to insufficient memory for accommodating these automatically created indexes.)

If you are importing tables from an existing database using the output of mysqldump, you can open the SQL script in a text editor and add the ENGINE option to any table creation statements, or replace any existing ENGINE options. Suppose that you have the world sample database on another MySQL server that does not support NDB Cluster, and you want to export the City table:

```
shell> mysqldump --add-drop-table world City > city_table.sql
```

The resulting city_table.sql file contains this table creation statement (and the INSERT statements necessary to import the table data):

```sql
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=MyISAM DEFAULT CHARSET=latin1;
```

You need to make sure that MySQL uses the NDBCLUSTER storage engine for this table. There are two ways that this can be accomplished. One of these is to modify the table definition before importing it into the Cluster database. Using the City table as an example, modify the ENGINE option of the definition as follows:

```sql
DROP TABLE IF EXISTS `City`;
CREATE TABLE `City` (
  `ID` int(11) NOT NULL auto_increment,
  `Name` char(35) NOT NULL default '',
  `CountryCode` char(3) NOT NULL default '',
  `District` char(20) NOT NULL default '',
  `Population` int(11) NOT NULL default '0',
  PRIMARY KEY  (`ID`)
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

This must be done for the definition of each table that is to be part of the clustered database. The easiest way to accomplish this is to do a search-and-replace on the file that contains the definitions and replace all instances of TYPE=engine_name or ENGINE=engine_name with ENGINE=NDBCLUSTER. If you do not want to modify the file, you can use the unmodified file to create the tables, and then use ALTER TABLE to change their storage engine. The particulars are given later in this section.

Assuming that you have already created a database named world on the SQL node of the cluster, you can then use the mysql command-line client to read city_table.sql, and create and populate the corresponding table in the usual manner:

```
shell> mysql world < city_table.sql
```

It is very important to keep in mind that the preceding command must be executed on the host where the SQL node is running (in this case, on the machine with the IP address **198.51.100.20**).
To create a copy of the entire world database on the SQL node, use mysqldump on the noncluster server to export the database to a file named world.sql (for example, in the /tmp directory). Then modify the table definitions as just described and import the file into the SQL node of the cluster like this:

```
shell> mysql world < /tmp/world.sql
```

If you save the file to a different location, adjust the preceding instructions accordingly.

Running SELECT queries on the SQL node is no different from running them on any other instance of a MySQL server. To run queries from the command line, you first need to log in to the MySQL Monitor in the usual way (specify the root password at the Enter password: prompt):

```
shell> mysql -u root -p
Enter password:
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.24-ndb-8.0.24
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
```

We simply use the MySQL server's root account and assume that you have followed the standard security precautions for installing a MySQL server, including setting a strong root password. For more information, see Securing the Initial MySQL Account.

It is worth taking into account that NDB Cluster nodes do not make use of the MySQL privilege system when accessing one another. Setting or changing MySQL user accounts (including the root account) affects only applications that access the SQL node, not interaction between nodes. See Section 7.17.2, "NDB Cluster and MySQL Privileges", for more information.

If you did not modify the ENGINE clauses in the table definitions prior to importing the SQL script, you should run the following statements at this point:

```
mysql> USE world;
mysql> ALTER TABLE City ENGINE=NDBCLUSTER;
mysql> ALTER TABLE Country ENGINE=NDBCLUSTER;
mysql> ALTER TABLE CountryLanguage ENGINE=NDBCLUSTER;
```

Selecting a database and running a SELECT query against a table in that database is also accomplished in the usual manner, as is exiting the MySQL Monitor:

```
mysql> USE world;
mysql> SELECT Name, Population FROM City ORDER BY Population DESC LIMIT 5;
```

```
+-----------+------------+
| Name      | Population |
+-----------+------------+
| Bombay    |   10500000 |
| Seoul     |    9981619 |
| São Paulo |    9968485 |
| Shanghai  |    9696300 |
| Jakarta   |    9604900 |
+-----------+------------+
```

```
mysql> \q
Bye
shell>
```

Applications that use MySQL can employ standard APIs to access NDB tables. It is important to remember that your application must access the SQL node, and not the management or data nodes. This brief example shows how we might execute the SELECT statement just shown by using the PHP 5.X mysqli extension running on a Web server elsewhere on the network:

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01 Transitional//EN"
"http://www.w3.org/TR/html4/loose.dtd">
<html>
<head>
<meta http-equiv="Content-Type"
"text/html; charset=iso-8859-1">
```
4.6 Safe Shutdown and Restart of NDB Cluster

To shut down the cluster, enter the following command in a shell on the machine hosting the management node:

```
shell> ndb_mgm -e shutdown
```

The `--e` option here is used to pass a command to the `ndb_mgm` client from the shell. (See Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, for more information about this option.) The command causes the `ndb_mgm`, `ndb_mgmd`, and any `ndbd` or `ndbmtd` processes to terminate gracefully. Any SQL nodes can be terminated using `mysqladmin shutdown` and other means. On Windows platforms, assuming that you have installed the SQL node as a Windows service, you can use `SC STOP service_name` or `NET STOP service_name`.

To restart the cluster on Unix platforms, run these commands:

```
Upgrading and Downgrading NDB Cluster

4.7 Upgrading and Downgrading NDB Cluster

This section provides information about NDB Cluster software and table file compatibility between different NDB Cluster 8.0 releases with regard to performing upgrades and downgrades as well as compatibility matrices and notes. You should already be familiar with installing and configuring NDB Cluster prior to attempting an upgrade or downgrade. See Chapter 5, Configuration of NDB Cluster.

Schema operations, including SQL DDL statements, cannot be performed while any data nodes are restarting, and thus during an online upgrade or downgrade of the cluster. For other information regarding the rolling restart procedure used to perform an online upgrade, see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”.

Important

Only compatibility between MySQL versions with regard to NDBCLUSTER is taken into account in this section, and there are likely other issues to be considered. As with any other MySQL software upgrade or downgrade, you are strongly encouraged to review the relevant portions of the MySQL Manual for the MySQL versions from which and to which you intend to migrate, before attempting an upgrade or downgrade of the NDB Cluster software. See Upgrading MySQL.
Upgrades and Downgrades, NDB Cluster 8.0

The table shown here provides information on NDB Cluster upgrade and downgrade compatibility among different releases of NDB 8.0. Additional notes about upgrades and downgrades to, from, or within the NDB Cluster 8.0 release series can be found following the table.

Upgrades and Downgrades, NDB Cluster 8.0

Figure 4.2 NDB Cluster Upgrade and Downgrade Compatibility, MySQL NDB Cluster 8.0

Version support. The following versions of NDB Cluster are supported for upgrades to GA releases of NDB Cluster 8.0 (8.0.19 and later):

- NDB Cluster 7.6: NDB 7.6.4 and later
- NDB Cluster 7.5: NDB 7.5.4 and later
- NDB Cluster 7.4: NDB 7.4.6 and later

To upgrade from a release series previous to NDB 7.4, you must upgrade in stages, first to one of the versions just listed, and then from that version to the latest NDB 8.0 release. In such cases, upgrading to the latest NDB 7.6 release is recommended as the first step.

Known Issues. The following issues are known to occur when upgrading to or between NDB 8.0 releases:

- Online downgrades from NDB 8.0.14 to previous releases are not supported. Tables created in NDB 8.0.14 are not backwards compatible with previous releases. This is due to a change in usage of the extra metadata property implemented by NDB tables to provide full support for the MySQL data dictionary.

  For more information, see Changes in NDB table extra metadata. See also MySQL Data Dictionary.

- Distributed privileges shared between MySQL servers as implemented in prior release series (see Distributed Privileges Using Shared Grant Tables) are not supported in NDB Cluster 8.0. When started, the mysqld supplied with NDB 8.0.16 and later checks for the existence of any grant tables
which use the NDB storage engine; if it finds any, it creates local copies (“shadow tables”) of these using InnoDB. This is true for each MySQL server connected to NDB Cluster. After this has been performed on all MySQL servers acting as NDB Cluster SQL nodes, the NDB grant tables may be safely removed using the `ndb_drop_table` utility supplied with the NDB Cluster distribution, like this:

```
ndb_drop_table -d mysql user db columns_priv tables_priv proxies_priv procs_priv
```

It is safe to retain the NDB grant tables, but they are not used for access control and are effectively ignored.

For more information about the MySQL privileges system used in NDB 8.0, see Section 7.12, “Distributed MySQL Privileges with NDB_STORED_USER”, as well as Grant Tables.

- In NDB 8.0.18, the binary configuration file format has been enhanced to provide support for greater numbers of nodes than in previous versions. The new format is not accessible to 8.0.17 and older nodes, although newer management servers can detect older nodes and communicate with them using the appropriate format.

Upgrades to NDB 8.0.18 or later from 8.0.17 and earlier should not be problematic in this regard. In the case of downgrades from NDB 8.0.18 or later to 8.0.17 or earlier, because older management servers cannot read the newer binary configuration file format, some manual intervention is required. When performing such a downgrade, it is necessary to remove any cached binary configuration files prior to starting the management using the older NDB software version, and to have the plaintext configuration file available for the management server to read. Alternatively, you can start the older management server using the `--initial` option (again, it is necessary to have the config.ini available). If the cluster uses multiple management servers, one of these two things must be done for each management server binary.

Also in connection with support for increased numbers of nodes, due to incompatible changes implemented in NDB 8.0.18 in the data node LCP Sysfile, it is necessary, when performing an online downgrade from NDB 8.0.18 (or later) to any prior release, to restart all data nodes using the `--initial` option.

Restarting the data nodes with `--initial` is also required when upgrading any release prior to NDB 7.6.4 to any NDB 8.0 release.

- Direct downgrades of clusters running more than 48 data nodes, or with data nodes using node IDs greater than 48, to NDB versions 8.0.17 and earlier from NDB 8.0.18 or later are not supported. It is necessary in such cases to reduce the number of data nodes, change the configurations for all data nodes such that they use node IDs less than or equal to 48, or both, as required not to exceed the old maximums.

- If you are downgrading from NDB 8.0 to NDB 7.5 or NDB 7.4, you must set an explicit value for `IndexMemory` in the cluster configuration file if none is already present. This is because NDB 8.0 does not use this parameter (which was removed in NDB 7.6) and sets it to 0 by default, whereas it is required in NDB 7.5 and NDB 7.4, in both of which the cluster refuses to start with `Invalid configuration received from Management Server...` if `IndexMemory` is not set to a nonzero value.

Setting `IndexMemory` is not required for downgrades from NDB 8.0 to NDB 7.6.

- NDB 8.0.22 adds support for IPv6 addressing for management nodes and data nodes in the config.ini file. To begin using IPv6 addresses as part of an upgrade, perform the following steps:
  1. Perform an upgrade of the cluster to version 8.0.22 or a later version of the NDB Cluster software in the usual manner.
  2. Change the addresses used in the config.ini file to IPv6 addresses.
  3. Perform a system restart of the cluster.
4.8 The NDB Cluster Auto-Installer (No longer supported)

Note
This feature has been removed from NDB Cluster, and is no longer supported. See Section 3.4, "What is New in NDB Cluster", for more information.

This section describes the web-based graphical configuration installer included as part of the NDB Cluster distribution. Topics discussed include an overview of the installer and its parts, software and other requirements for running the installer, navigating the GUI, and using the installer to set up and start or stop an NDB Cluster on one or more host computers.

The NDB Cluster Auto-Installer is made up of two components. The front end is a GUI client implemented as a Web page that loads and runs in a standard Web browser such as Firefox or Microsoft Internet Explorer. The back end is a server process (ndb_setup.py) that runs on the local machine or on another host to which you have access.

These two components (client and server) communicate with each other using standard HTTP requests and responses. The back end can manage NDB Cluster software programs on any host where the back end user has granted access. If the NDB Cluster software is on a different host, the back end relies on SSH for access.

4.8.1 NDB Cluster Auto-Installer Requirements

This section provides information on supported operating platforms and software, required software, and other prerequisites for running the NDB Cluster Auto-Installer.

Supported platforms. The NDB Cluster Auto-Installer is available with NDB 8.0 distributions for recent versions of Linux, Windows, Solaris, and macOS. For more detailed information about platform support for NDB Cluster and the NDB Cluster Auto-Installer, see https://www.mysql.com/support/supportedplatforms/cluster.html.

Supported web browsers. The web-based installer is supported with recent versions of Firefox and Microsoft Internet Explorer. It should also work with recent versions of Opera, Safari, and Chrome, although we have not thoroughly tested for compatibility with these browsers.

Required software—setup host. The following software must be installed on the host where the Auto-Installer is run:

- **Python 2.6 or higher.** The Auto-Installer requires the Python interpreter and standard libraries. If these are not already installed on the system, you may be able to add them using the system's package manager. Otherwise, you can download them from http://python.org/download/.

- **Paramiko 2 or higher.** You can download this from http://www.lag.net/paramiko/ if it is not available from your system's package manager.

- **Pycrypto version 1.9 or higher.** This cryptography module is required by Paramiko, and can be installed using `pip install cryptography`. If `pip` is not installed, and the module is not available using your system's package manager, you can download it from https://www.dlitz.net/software/pycrypto/.

All of the software in the preceding list is included in the Windows version of the configuration tool, and does not need to be installed separately.

Required software—remote hosts. The only software required for remote hosts where you wish to deploy NDB Cluster nodes is the SSH server, which is usually installed by default on Linux and Solaris systems. Several alternatives are available for Windows; for an overview of these, see http://en.wikipedia.org/wiki/Comparison_of_SSH_servers.
An additional requirement when using multiple hosts is that it is possible to authenticate to any of the remote hosts using SSH and the proper keys or user credentials, as discussed in the next few paragraphs:

**Authentication and security.** Three basic security or authentication mechanisms for remote access are available to the Auto-Installer, which we list and describe here:

- **SSH.** A secure shell connection is used to enable the back end to perform actions on remote hosts. For this reason, an SSH server must be running on the remote host. In addition, the operating system user running the installer must have access to the remote server, either with a user name and password, or by using public and private keys.

  **Important**
  
  You should never use the system root account for remote access, as this is extremely insecure. In addition, mysql cannot normally be started by system root. For these and other reasons, you should provide SSH credentials for a regular user account on the target system, and not for system root. For more information about this issue, see How to Run MySQL as a Normal User.

- **HTTPS.** Remote communication between the Web browser front end and the back end is not encrypted by default, which means that information such as the user's SSH password is transmitted as cleartext that is readable to anyone. For communication from a remote client to be encrypted, the back end must have a certificate, and the front end must communicate with the back end using HTTPS rather than HTTP. Enabling HTTPS is accomplished most easily through issuing a self-signed certificate. Once the certificate is issued, you must make sure that it is used. You can do this by starting ndb_setup.py from the command line with the --use-https (-S) and --cert-file (-c) options.

  A sample certificate file cfg.pem is included and is used by default. This file is located in the mcc directory under the installation share directory; on Linux, the full path to the file is normally /usr/share/mysql/mcc/cfg.pem. On Windows systems, this is usually C:\Program Files\MySQL\MySQL Server 8.0\share\mcc\cfg.pem. Letting the default be used means that, for testing purposes, you can simply start the installer with the -S option to use an HTTPS connection between the browser and the back end.

  The Auto-Installer saves the configuration file for a given cluster mycluster01 as mycluster01.mcc in the home directory of the user invoking the ndb_setup.py executable. This file is encrypted with a passphrase supplied by the user (using Fernet); because HTTP transmits the passphrase in the clear, it is strongly recommended that you always use an HTTPS connection to access the Auto-Installer on a remote host.

- **Certificate-based authentication.** The back end ndb_setup.py process can execute commands on the local host as well as remote hosts. This means that anyone connecting to the back end can take charge of how commands are executed. To reject unwanted connections to the back end, a certificate may be required for authentication of the client. In this case, a certificate must be issued by the user, installed in the browser, and made available to the back end for authentication purposes. You can enact this requirement (together with or in place of password or key authentication) by starting ndb_setup.py with the --ca-cert-file (-a) option.

  There is no need or requirement for secure authentication when the client browser is running on the same host as the Auto-Installer back end.

  See also Section 7.17, “NDB Cluster Security Issues”, which discusses security considerations to take into account when deploying NDB Cluster, as well as Security, for more general MySQL security information.

### 4.8.2 Using the NDB Cluster Auto-Installer
The NDB Cluster Auto-Installer interface is made up of several pages, each corresponding to a step in the process used to configure and deploy an NDB Cluster. These pages are listed here, in order:

- **Welcome**: Begin using the Auto-Installer by choosing either to configure a new NDB Cluster, or to continue configuring an existing one.
- **Define Cluster**: Set basic information about the cluster as a whole, such as name, hosts, and load type. Here you can also set the SSH authentication type for accessing remote hosts, if needed.
- **Define Hosts**: Identify the hosts where you intend to run NDB Cluster processes.
- **Define Processes**: Assign one or more processes of a given type or types to each cluster host.
- **Define Parameters**: Set configuration attributes for processes or types of processes.
- **Deploy Configuration**: Deploy the cluster with the configuration set previously; start and stop the deployed cluster.

### NDB Cluster Installer Settings and Help Menus

These menus are shown on all screens except for the Welcome screen. They provide access to installer settings and information. The Settings menu is shown here in more detail:

**Figure 4.3 NDB Cluster Auto-Installer Settings menu**

The Settings menu has the following entries:

- **Automatically save configuration as cookies**: Save your configuration information—such as host names, process data, and parameter values—as a cookie in the browser. When this option is chosen, all information except any SSH password is saved. This means that you can quit and restart the browser, and continue working on the same configuration from where you left off at the end of the previous session. This option is enabled by default.

  The SSH password is never saved; if you use one, you must supply it at the beginning of each new session.

- **Show advanced configuration options**: Shows by default advanced configuration parameters where available.

  Once set, the advanced parameters continue to be used in the configuration file until they are explicitly changed or reset. This is regardless of whether the advanced parameters are currently visible in the installer; in other words, disabling the menu item does not reset the values of any of these parameters.

  You can also toggle the display of advanced parameters for individual processes on the Define Parameters screen.
Using the NDB Cluster Auto-Installer

This option is disabled by default.

- **Automatically get resource information for new hosts**: Query new hosts automatically for hardware resource information to pre-populate a number of configuration options and values. In this case, the suggested values are not mandatory, but they are used unless explicitly changed using the appropriate editing options in the installer.

  This option is enabled by default.

The installer **Help** menu is shown here:

**Figure 4.4 NDB Cluster Auto-Installer Help menu**

The **Help** menu provides several options, described in the following list:

- **Contents**: Show the built-in user guide. This is opened in a separate browser window, so that it can be used simultaneously with the installer without interrupting workflow.

- **Current page**: Open the built-in user guide to the section describing the page currently displayed in the installer.

- **About**: open a dialog displaying the installer name and the version number of the NDB Cluster distribution with which it was supplied.

The Auto-Installer also provides context-sensitive help in the form of tooltips for most input widgets.

In addition, the names of most NDB configuration parameters are linked to their descriptions in the online documentation. The documentation is displayed in a separate browser window.

The next section discusses starting the Auto-Installer. The sections immediately following it describe in greater detail the purpose and function of each of these pages in the order listed previously.

**Starting the NDB Cluster Auto-Installer**

The Auto-Installer is provided together with the NDB Cluster software. Separate RPM and `.deb` packages containing only the Auto-Installer are also available for many Linux distributions. (See Chapter 4, *NDB Cluster Installation*.)

The present section explains how to start the installer. You can do by invoking the `ndb_setup.py` executable.

**User and privileges**

You should run the `ndb_setup.py` as a normal user; no special privileges are needed to do so. You should *not* run this program as the `mysql` user, or using
the system root or Administrator account; doing so may cause the installation to fail.

ndb_setup.py is found in the bin within the NDB Cluster installation directory; a typical location might be /usr/local/mysql/bin on a Linux system or C:\Program Files\MySQL\MySQL Server 8.0\bin on a Windows system. This can vary according to where the NDB Cluster software is installed on your system, and the installation method.

On Windows, you can also start the installer by running setup.bat in the NDB Cluster installation directory. When invoked from the command line, this batch file accepts the same options as ndb_setup.py.

ndb_setup.py can be started with any of several options that affect its operation, but it is usually sufficient to allow the default settings be used, in which case you can start ndb_setup.py by either of the following two methods:

1. Navigate to the NDB Cluster bin directory in a terminal and invoke it from the command line, without any additional arguments or options, like this:

```
shell> ndb_setup.py
Running out of install dir: /usr/local/mysql/bin
Starting web server on port 8081
URL is https://localhost:8081/welcome.html
deathkey=627876
Press CTRL+C to stop web server.
The application should now be running in your browser.
(Alternatively you can navigate to https://localhost:8081/welcome.html to start it)
```

This works regardless of operating platform.

2. Navigate to the NDB Cluster bin directory in a file browser (such as Windows Explorer on Windows, or Konqueror, Dolphin, or Nautilus on Linux) and activate (usually by double-clicking) the ndb_setup.py file icon. This works on Windows, and should work with most common Linux desktops as well.

On Windows, you can also navigate to the NDB Cluster installation directory and activate the setup.bat file icon.

In either case, once ndb_setup.py is invoked, the Auto-Installer’s Welcome screen should open in the system’s default web browser. If not, you should be able to open the page http://localhost:8081/welcome.html or https://localhost:8081/welcome.html manually in the browser.

In some cases, you may wish to use non-default settings for the installer, such as specifying HTTPS for connections, or a different port for the Auto-Installer’s included web server to run on, in which case you must invoke ndb_setup.py with one or more startup options with values overriding the necessary defaults. The same startup options can be used on Windows systems with the setup.bat file supplied for such platforms in the NDB Cluster software distribution. This can be done using the command line, but if you want or need to start the installer from a desktop or file browser while employing one or more of these options, it is also possible to create a script or batch file containing the proper invocation, then to double-click its file icon in the file browser to start the installer. (On Linux systems, you might also need to make the script file executable first.) If you plan to use the Auto-Installer from a remote host, you should start using the -S option. For information about this and other advanced startup options for the NDB Cluster Auto-Installer, see Section 6.26, “ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)”.

NDB Cluster Auto-Installer Welcome Screen

The Welcome screen is loaded in the default browser when ndb_setup.py is invoked. The first time the Auto-Installer is run (or if for some other reason there are no existing configurations), this screen appears as shown here:
In this case, the only choice of cluster listed is for configuration of a new cluster, and both the View Cfg and Continue buttons are inactive.

To create a new configuration, enter and confirm a passphrase in the text boxes provided. When this has been done, you can click Continue to proceed to the Define Cluster screen where you can assign a name to the new cluster.

If you have previously created one or more clusters with the Auto-Installer, they are listed by name. This example shows an existing cluster named mycluster-1:
To view the configuration for and work with a given cluster, select the radiobutton next to its name in the list, then enter and confirm the passphrase that was used to create it. When you have done this correctly, you can click View Cfg to view and edit this cluster’s configuration.

**NDB Cluster Auto-Installer Define Cluster Screen**

The Define Cluster screen is appears following the Welcome screen, and is used for setting general properties of the cluster. The layout of the Define Cluster screen is shown here:

**Figure 4.7 The NDB Cluster Auto-Installer Define Cluster screen**

This screen and subsequent screens also include Settings and Help menus which are described later in this section; see NDB Cluster Installer Settings and Help Menus.

The Define Cluster screen allows you to set three sorts of properties for the cluster: cluster properties, SSH properties, and installation properties.

Cluster properties that can be set on this screen are listed here:

- **Cluster name**: A name that identifies the cluster; in this example, this is `mycluster-1`. The name is set on the previous screen and cannot be changed here.
- **Host list**: A comma-delimited list of one or more hosts where cluster processes should run. By default, this is `127.0.0.1`. If you add remote hosts to the list, you must be able to connect to them using the credentials supplied as SSH properties.
- **Application type**: Choose one of the following:
  1. **Simple testing**: Minimal resource usage for small-scale testing. This the default. *Not intended for production environments.*
  2. **Web**: Maximize performance for the given hardware.
  3. **Real-time**: Maximize performance while maximizing sensitivity to timeouts in order to minimize the time needed to detect failed cluster processes.
- **Write load**: Choose a level for the anticipated number of writes for the cluster as a whole. You can choose any one of the following levels:
1. **Low:** The expected load includes fewer than 100 write transactions per second.

2. **Medium:** The expected load includes 100 to 1000 write transactions per second; this is the default.

3. **High:** The expected load includes more than 1000 write transactions per second.

SSH properties are described in the following list:

- **Key-Based SSH:** Check this box to use key-enabled login to the remote host. If checked, the key user and passphrase must also be supplied; otherwise, a user and password for a remote login account are needed.

- **User:** Name of user with remote login access.

- **Password:** Password for remote user.

- **Key user:** Name of the user for whom the key is valid, if not the same as the operating system user.

- **Key passphrase:** Passphrase for the key, if required.

- **Key file:** Path to the key file. The default is `~/.ssh/id_rsa`.

The SSH properties set on this page apply to all hosts in the cluster. They can be overridden for a given host by editing that host's properties on the **Define Hosts** screen.

Two installation properties can also be set on this screen:

- **Install MySQL Cluster:** This setting determines the source from which the Auto-Installer installs NDB Cluster software, if any, on the cluster hosts. Possible values and their effects are listed here:

  1. **DOCKER:** Try to install the MySQL Cluster Docker image from [https://hub.docker.com/r/mysql/mysql-cluster/](https://hub.docker.com/r/mysql/mysql-cluster/) on each host

  2. **REPO:** Try to install the NDB Cluster software from the [MySQL Repositories](https://dev.mysql.com/downloads/released/ndb/) on each host

  3. **BOTH:** Try to install either the Docker image or the software from the repository on each host, giving preference to the repository

  4. **NONE:** Do not install the NDB Cluster software on the hosts; this is the default

- **Open FW Ports:** Check this check box to have the installer attempt to open ports required by NDB Cluster processes on all hosts.

The next figure shows the **Define Cluster** page with settings for a small test cluster with all nodes running on **localhost**: 
Using the NDB Cluster Auto-Installer

**Figure 4.8 The NDB Cluster Auto-Installer Define Cluster screen, with settings for a test cluster**

After making the desired settings, you can save them to the configuration file and proceed to the **Define Hosts** screen by clicking the **Save & Next** button.

If you exit the installer without saving, no changes are made to the configuration file.

**NDB Cluster Auto-Installer Define Hosts Screen**

The **Define Hosts** screen, shown here, provides a means of viewing and specifying several key properties of each cluster host:
Properties shown include the following:

- **Host**: Name or IP address of this host
- **Res.info**: Shows OK if the installer was able to retrieve requested resource information from this host
- **Platform**: Operating system or platform
- **Memory (MB)**: Amount of RAM on this host
- **Cores**: Number of CPU cores available on this host
- **MySQL Cluster install directory**: Path to directory where the NDB Cluster software is installed on this host; defaults to /usr/local/bin
- **MySQL Cluster data directory**: Path to directory used for data by NDB Cluster processes on this host; defaults to /var/lib/mysql-cluster.
- **DiskFree**: Free disk space in bytes

For hosts with multiple disks, only the space available on the disk used for the data directory is shown.

This screen also provides an extended view for each host that includes the following properties:

- **FDQN**: This host's fully qualified domain name, used by the installer to connect with it, distribute configuration information to it, and start and stop cluster processes on it.
- **Internal IP**: The IP address used for communication with cluster processes running on this host by processes running elsewhere.
- **OS Details**: Detailed operating system name and version information.
- **Open FW**: If this check box is enabled, the installer attempts to open ports in the host's firewall needed by cluster processes.
- **REPO URL**: URL for MySQL NDB Cluster repository
- **DOCKER URL**: URL for MySQL NDB Cluster Docker images; for NDB 8.0, this is mysql/mysql-cluster:8.0.
- **Install**: If this check box is enabled, the Auto-Installer attempts to install the NDB Cluster software on this host.
The extended view is shown here:

**Figure 4.10 NDB Cluster Define Hosts screen, extended host info view**

All cells in the display are editable, with the exceptions of those in the **Host**, **Res.info**, and **FQDN** columns.

Be aware that it may take some time for information to be retrieved from remote hosts. Fields for which no value could be retrieved are indicated with an ellipsis (...). You can retry the fetching of resource information from one or more hosts by selecting the hosts in the list and then clicking the **Refresh selected host(s)** button.

**Adding and Removing Hosts**

You can add one or more hosts by clicking the **Add Host** button and entering the required properties where indicated in the **Add new host** dialog, shown here:
This dialog includes the following fields:

- **Host name**: A comma-separated list of one or more host names, IP addresses, or both. These must be accessible from the host where the Auto-Installer is running.

- **Host internal IP (VPN)**: If you are setting up the cluster to run on a VPN or other internal network, enter the IP address or addresses used for contact by cluster nodes on other hosts.
• **Key-based auth**: If checked, enables key-based authentication. You can enter any additional needed information in the **User**, **Passphrase**, and **Key file** fields.

• **Ordinary login**: If accessing this host using a password-based login, enter the appropriate information in the **User** and **Password** fields.

• **Open FW ports**: Selecting this check box allows the installer try opening any ports needed by cluster processes in this host's firewall.

• **Configure installation**: Checking this allows the Auto-Install to attempt to set up the NDB Cluster software on this host.

To save the new host and its properties, click **Add**. If you wish to cancel without saving any changes, click **Cancel** instead.

Similarly, you can remove one or more hosts using the button labelled **Remove selected host(s)**. **When you remove a host, any process which was configured for that host is also removed.**

**Warning**

**Remove selected host(s)** acts immediately. There is no confirmation dialog. If you remove a host in error, you must re-enter its name and properties manually using **Add host**.

If the SSH user credentials on the **Define Cluster screen** are changed, the Auto-Installer attempts to refresh the resource information from any hosts for which information is missing.

You can edit the host's platform name, hardware resource information, installation directory, and data directory by clicking the corresponding cell in the grid, by selecting one or more hosts and clicking the button labelled **Edit selected host(s)**. This causes a dialog box to appear, in which these fields can be edited, as shown here:

**Figure 4.12 NDB Cluster Auto-Installer Edit Hosts dialog**

When more than one host is selected, any edited values are applied to all selected hosts.
Once you have entered all desired host information, you can use the Save & Next button to save the information to the cluster's configuration file and proceed to the Define Processes screen, where you can set up NDB Cluster processes on one or more hosts.

**NDB Cluster Auto-Installer Define Processes Screen**

The Define Processes screen, shown here, provides a way to assign NDB Cluster processes (nodes) to cluster hosts:

**Figure 4.13 NDB Cluster Auto-Installer Define Processes dialog**

This screen contains a process tree showing cluster hosts and processes set up to run on each one, as well as a panel which displays information about the item currently selected in the tree.

When this screen is accessed for the first time for a given cluster, a default set of processes is defined for you, based on the number of hosts. If you later return to the Define Hosts screen, remove all hosts, and add new hosts, this also causes a new default set of processes to be defined.

NDB Cluster processes are of the types described in this list:

- **Management node.** Performs administrative tasks such as stopping individual data nodes, querying node and cluster status, and making backups. Executable: ndb_mgmd.
- **Single-threaded data node.** Stores data and executes queries. Executable: ndbd.
- **Multi-threaded data node.** Stores data and executes queries with multiple worker threads executing in parallel. Executable: ndbmt.d.
- **SQL node.** MySQL server for executing SQL queries against NDB. Executable: mysqld.
- **API node.** A client accessing data in NDB by means of the NDB API or other low-level client API, rather than by using SQL. See MySQL NDB Cluster API Developer Guide, for more information.

For more information about process (node) types, see Section 3.1, “NDB Cluster Core Concepts”.

Processes shown in the tree are numbered sequentially by type, for each host—for example, SQL node 1, SQL node 2, and so on—to simplify identification.
Each management node, data node, or SQL process must be assigned to a specific host, and is not allowed to run on any other host. An API node may be assigned to a single host, but this is not required. Instead, you can assign it to the special **Any host** entry which the tree also contains in addition to any other hosts, and which acts as a placeholder for processes that are allowed to run on any host. **Only API processes may use this Any host entry.**

**Adding processes.** To add a new process to a given host, either right-click that host's entry in the tree, then select the **Add process** popup when it appears, or select a host in the process tree, and press the **Add process** button below the process tree. Performing either of these actions opens the add process dialog, as shown here:

![Figure 4.14 NDB Cluster Auto-Installer Add Process Dialog](image)

Here you can select from among the available process types described earlier this section; you can also enter an arbitrary process name to take the place of the suggested value, if desired.

**Removing processes.** To delete a process, select that process in the tree and use the **Del process** button.

When you select a process in the process tree, information about that process is displayed in the information panel, where you can change the process name and possibly its type. You can change a multi-threaded data node (**ndbmt.c** to a single-threaded data node (**ndbd**), or the reverse, only; no other process type changes are allowed. **If you want to make a change between any other process types, you must delete the original process first, then add a new process of the desired type.**

**NDB Cluster Auto-Installer Define Parameters Screen**

Like the **Define Processes** screen, this screen includes a process tree; the **Define Parameters** process tree is organized by process or node type, in groups labelled **Management Layer**, **Data Layer**, **SQL Layer**, and **API Layer**. An information panel displays information regarding the item currently selected. The **Define Attributes** screen is shown here:
The check box labelled **Show advanced configuration**, when checked, makes advanced options for data node and SQL node processes visible in the information pane. These options are set and used whether or not they are visible. You can also enable this behavior globally by checking **Show advanced configuration options** under **Settings** (see NDB Cluster Installer Settings and Help Menus).

You can edit attributes for a single process by selecting that process from the tree, or for all processes of the same type in the cluster by selecting one of the **Layer** folders. A per-attribute value set for a given attribute overrides any per-group setting for that attribute that would otherwise apply to the process in question. An example of such an information panel (for an SQL process) is shown here:
Attributes whose values can be overridden are shown in the information panel with a button bearing a plus sign. This + button activates an input widget for the attribute, enabling you to change its value. When the value has been overridden, this button changes into a button showing an X. The X button undoes any changes made to a given attribute, which immediately reverts to the predefined value.

All configuration attributes have predefined values calculated by the installer, based such factors as host name, node ID, node type, and so on. In most cases, these values may be left as they are. If you are not familiar with it already, it is highly recommended that you read the applicable documentation before making changes to any of the attribute values. To make finding this information easier, each attribute name shown in the information panel is linked to its description in the online NDB Cluster documentation.

**NDB Cluster Auto-Installer Deploy Configuration Screen**

This screen allows you to perform the following tasks:

- Review process startup commands and configuration files to be applied
- Distribute configuration files by creating any necessary files and directories on all cluster hosts—that is, `deploy` the cluster as presently configured
- Start and stop the cluster

The **Deploy Configuration** screen is shown here:
Like the Define Parameters screen, this screen features a process tree which is organized by process type. Next to each process in the tree is a status icon indicating the current status of the process: connected (CONNECTED), starting (STARTING), running (STARTED), stopping (STOPPING), or disconnected (NO_CONTACT). The icon shows green if the process is connected or running; yellow if it is starting or stopping; red if the process is stopped or cannot be contacted by the management server.

This screen also contains two information panels, one showing the startup command or commands needed to start the selected process. (For some processes, more than one command may be required—for example, if initialization is necessary.) The other panel shows the contents of the configuration file, if any, for the given process.

This screen also contains four buttons, labelled as and performing the functions described in the following list:

- **Install cluster**: Nonfunctional in this release; implementation intended for a future release.

- **Deploy cluster**: Verify that the configuration is valid. Create any directories required on the cluster hosts, and distribute the configuration files onto the hosts. A progress bar shows how far the deployment has proceeded, as shown here, and a dialog is displayed when the deployment has completed, as shown here:
Figure 4.18 Cluster Deployment Process
Using the NDB Cluster Auto-Installer

• **Start cluster:** The cluster is deployed as with **Deploy cluster**, after which all cluster processes are started in the correct order.

Starting these processes may take some time. If the estimated time to completion is too large, the installer provides an opportunity to cancel or to continue of the startup procedure. A progress bar indicates the current status of the startup procedure, as shown here:

**Figure 4.19 Cluster Startup Process with Progress Bar**

The process status icons next to the items shown in the process tree also update with the status of each process.

A confirmation dialog is shown when the startup process has completed, as shown here:

**Figure 4.20 Cluster Startup, Process Completed Dialog**

• **Stop cluster:** After the cluster has been started, you can stop it using this. As with starting the cluster, cluster shutdown is not instantaneous, and may require some time complete. A progress bar, similar to that displayed during cluster startup, shows the approximate current status of the cluster.
shutdown procedure, as do the process status icons adjoining the process tree. The progress bar is shown here:

**Figure 4.21 Cluster Shutdown Process, with Progress Bar**

A confirmation dialog indicates when the shutdown process is complete:

**Figure 4.22 Cluster Shutdown, Process Completed Dialog**

The Auto-Installer generates a `config.ini` file containing NDB node parameters for each management node, as well as a `my.cnf` file containing the appropriate options for each `mysqld` process in the cluster. No configuration files are created for data nodes or API nodes.
Chapter 5 Configuration of NDB Cluster

Table of Contents

5.1 Quick Test Setup of NDB Cluster ................................................................. 105
5.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables .............. 107
  5.2.1 NDB Cluster Data Node Configuration Parameters ...................................... 108
  5.2.2 NDB Cluster Management Node Configuration Parameters ............................. 108
  5.2.3 NDB Cluster SQL Node and API Node Configuration Parameters ................. 109
  5.2.4 Other NDB Cluster Configuration Parameters ............................................. 109
  5.2.5 NDB Cluster mysql Option and Variable Reference .................................... 109
5.3 NDB Cluster Configuration Files .................................................................... 110
  5.3.1 NDB Cluster Configuration: Basic Example .................................................. 111
  5.3.2 Recommended Starting Configuration for NDB Cluster ................................. 113
  5.3.3 NDB Cluster Connection Strings ................................................................ 116
  5.3.4 Defining Computers in an NDB Cluster ...................................................... 117
  5.3.5 Defining an NDB Cluster Management Server ............................................ 117
  5.3.6 Defining NDB Cluster Data Nodes ............................................................. 122
  5.3.7 Defining SQL and Other API Nodes in an NDB Cluster ............................... 177
  5.3.8 Defining the System ..................................................................................... 182
  5.3.9 MySQL Server Options and Variables for NDB Cluster ............................... 183
  5.3.10 NDB Cluster TCP/IP Connections ............................................................. 237
  5.3.11 NDB Cluster TCP/IP Connections Using Direct Connections ...................... 240
  5.3.12 NDB Cluster Shared-Memory Connections ................................................ 241
  5.3.13 Data Node Memory Management .............................................................. 244
  5.3.14 Configuring NDB Cluster Send Buffer Parameters .................................... 248
5.4 Using High-Speed Interconnects with NDB Cluster ........................................... 248

A MySQL server that is part of an NDB Cluster differs in one chief respect from a normal (nonclustered) MySQL server, in that it employs the NDB storage engine. This engine is also referred to sometimes as NDBCLUSTER, although NDB is preferred.

To avoid unnecessary allocation of resources, the server is configured by default with the NDB storage engine disabled. To enable NDB, you must modify the server’s my.cnf configuration file, or start the server with the --ndbcluster option.

This MySQL server is a part of the cluster, so it also must know how to access a management node to obtain the cluster configuration data. The default behavior is to look for the management node on localhost. However, should you need to specify that its location is elsewhere, this can be done in my.cnf, or with the mysql client. Before the NDB storage engine can be used, at least one management node must be operational, as well as any desired data nodes.

For more information about --ndbcluster and other mysql options specific to NDB Cluster, see Section 5.3.9.1, “MySQL Server Options for NDB Cluster”.

For general information about installing NDB Cluster, see Chapter 4, NDB Cluster Installation.

5.1 Quick Test Setup of NDB Cluster

To familiarize you with the basics, we describe the simplest possible configuration for a functional NDB Cluster. After this, you should be able to design your desired setup from the information provided in the other relevant sections of this chapter.

First, you need to create a configuration directory such as /var/lib/mysql-cluster, by executing the following command as the system root user:

```shell
mkdir /var/lib/mysql-cluster
```
In this directory, create a file named `config.ini` that contains the following information. Substitute appropriate values for `HostName` and `DataDir` as necessary for your system.

```ini
# file "config.ini" - showing minimal setup consisting of 1 data node,
# 1 management server, and 3 MySQL servers.
# The empty default sections are not required, and are shown only for
# the sake of completeness.
# Data nodes must provide a hostname but MySQL Servers are not required
to do so.
# If you don't know the hostname for your machine, use localhost.
# The DataDir parameter also has a default value, but it is recommended to
# set it explicitly.
# Note: [db], [api], and [mgm] are aliases for [ndbd], [mysqld], and [ndb_mgmd],
# respectively. [db] is deprecated and should not be used in new installations.

[ndbd default]
NoOfReplicas= 1

[mysqld  default]
[ndb_mgmd default]
[tcp default]
[ndb_mgmd]
HostName= myhost.example.com
[ndbd]
HostName= myhost.example.com
DataDir= /var/lib/mysql-cluster
[mysqld]
[mysqld]
[mysqld]
```

You can now start the `ndb_mgmd` management server. By default, it attempts to read the `config.ini` file in its current working directory, so change location into the directory where the file is located and then invoke `ndb_mgmd`:

```shell
cd /var/lib/mysql-cluster
ndb_mgmd
```

Then start a single data node by running `ndbd`:

```shell
ndbd
```

For command-line options which can be used when starting `ndbd`, see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

By default, `ndbd` looks for the management server at `localhost` on port 1186.

---

**Note**

If you have installed MySQL from a binary tarball, you must to specify the path of the `ndb_mgmd` and `ndbd` servers explicitly. (Normally, these can be found in `/usr/local/mysql/bin`.)

Finally, change location to the MySQL data directory (usually `/var/lib/mysql` or `/usr/local/mysql/data`), and make sure that the `my.cnf` file contains the option necessary to enable the NDB storage engine:

```ini
[mysqld]
ndbcluster
```

You can now start the MySQL server as usual:

```shell
mysql_safe --user=mysql &
```

Wait a moment to make sure the MySQL server is running properly. If you see the notice `mysql ended`, check the server's `.err` file to find out what went wrong.

If all has gone well so far, you now can start using the cluster. Connect to the server and verify that the `NDBCluster` storage engine is enabled:

```shell
mysql
```
Welcome to the MySQL monitor. Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 8.0.25
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>

SHOW ENGINES\G

*************************** 12. row ***************************
Engine: NDBCLUSTER
Support: YES
Comment: Clustered, fault-tolerant, memory-based tables

*************************** 13. row ***************************
Engine: NDB
Support: YES
Comment: Alias for NDBCLUSTER

The row numbers shown in the preceding example output may be different from those shown on your system, depending upon how your server is configured.

Try to create an NDBCLUSTER table:

shell> mysql
mysql> USE test;
Database changed
mysql> CREATE TABLE ctest (i INT) ENGINE=NDBCLUSTER;
Query OK, 0 rows affected (0.09 sec)
mysql>

SHOW CREATE TABLE ctest \G

*************************** 1. row ***************************
Table: ctest
Create Table: CREATE TABLE `ctest` (  
  `i` int(11) default NULL  ) ENGINE=ndbcluster DEFAULT CHARSET=latin1
1 row in set (0.00 sec)

To check that your nodes were set up properly, start the management client:

shell> ndb_mgm

Use the SHOW command from within the management client to obtain a report on the cluster's status:

ndb_mgm> SHOW
Cluster Configuration

[nedb (NDB)]  1 node(s)
id=2 @127.0.0.1 (Version: 8.0.24-ndb-8.0.24, Nodegroup: 0, *)
[ndb_mgmd (MGM)]  1 node(s)
id=1 @127.0.0.1 (Version: 8.0.24-ndb-8.0.24)
[mysqld (API)]  3 node(s)
id=3 @127.0.0.1 (Version: 8.0.24-ndb-8.0.24)
id=4 (not connected, accepting connect from any host)
id=5 (not connected, accepting connect from any host)

At this point, you have successfully set up a working NDB Cluster. You can now store data in the cluster by using any table created with ENGINE=NDBCLUSTER or its alias ENGINE=NDB.

5.2 Overview of NDB Cluster Configuration Parameters, Options, and Variables

The next several sections provide summary tables of NDB Cluster node configuration parameters used in the config.ini file to govern various aspects of node behavior, as well as of options and variables read by mysqld from a my.cnf file or from the command line when run as an NDB Cluster process. Each of the node parameter tables lists the parameters for a given type (ndbd, ndb_mgmd, mysqld, computer, tcp, or shm). All tables include the data type for the parameter, option, or variable, as well as its default, minimum, and maximum values as applicable.

Considerations when restarting nodes. For node parameters, these tables also indicate what type of restart is required (node restart or system restart)—and whether the restart must be done with
--initial—to change the value of a given configuration parameter. When performing a node restart or an initial node restart, all of the cluster's data nodes must be restarted in turn (also referred to as a rolling restart). It is possible to update cluster configuration parameters marked as node online—that is, without shutting down the cluster—in this fashion. An initial node restart requires restarting each ndbd process with the --initial option.

A system restart requires a complete shutdown and restart of the entire cluster. An initial system restart requires taking a backup of the cluster, wiping the cluster file system after shutdown, and then restoring from the backup following the restart.

In any cluster restart, all of the cluster's management servers must be restarted for them to read the updated configuration parameter values.

Important

Values for numeric cluster parameters can generally be increased without any problems, although it is advisable to do so progressively, making such adjustments in relatively small increments. Many of these can be increased online, using a rolling restart.

However, decreasing the values of such parameters—whether this is done using a node restart, node initial restart, or even a complete system restart of the cluster—is not to be undertaken lightly; it is recommended that you do so only after careful planning and testing. This is especially true with regard to those parameters that relate to memory usage and disk space, such as MaxNoOfTables, MaxNoOfOrderedIndexes, and MaxNoOfUniqueHashIndexes. In addition, it is the generally the case that configuration parameters relating to memory and disk usage can be raised using a simple node restart, but they require an initial node restart to be lowered.

Because some of these parameters can be used for configuring more than one type of cluster node, they may appear in more than one of the tables.

Note

4294967039 often appears as a maximum value in these tables. This value is defined in the NDBCLUSTER sources as MAX_INT_RNIL and is equal to 0xFFFFFEFF, or $2^{32} - 2^9 - 1$.

5.2.1 NDB Cluster Data Node Configuration Parameters

The listings in this section provide information about parameters used in the [ndbd] or [ndbd default] sections of a config.ini file for configuring NDB Cluster data nodes. For detailed descriptions and other additional information about each of these parameters, see Section 5.3.6, “Defining NDB Cluster Data Nodes”.

These parameters also apply to ndbmt, the multithreaded version of ndbd. For more information, see Section 6.3, “ndbmt — The NDB Cluster Data Node Daemon (Multi-Threaded)”.

The following parameters are specific to ndbmt:

5.2.2 NDB Cluster Management Node Configuration Parameters

The listing in this section provides information about parameters used in the [ndb_mgmd] or [mgm] section of a config.ini file for configuring NDB Cluster management nodes. For detailed descriptions and other additional information about each of these parameters, see Section 5.3.5, “Defining an NDB Cluster Management Server”.

108
5.2.3 NDB Cluster SQL Node and API Node Configuration Parameters

The listing in this section provides information about parameters used in the [mysqld] and [api] sections of a config.ini file for configuring NDB Cluster SQL nodes and API nodes. For detailed descriptions and other additional information about each of these parameters, see Section 5.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

For a discussion of MySQL server options for NDB Cluster, see Section 5.3.9.1, “MySQL Server Options for NDB Cluster”. For information about MySQL server system variables relating to NDB Cluster, see Section 5.3.9.2, “NDB Cluster System Variables”.

Note

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new [mysqld] or [api] sections to the config.ini file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is not necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

5.2.4 Other NDB Cluster Configuration Parameters

The listings in this section provide information about parameters used in the [computer], [tcp], and [shm] sections of a config.ini file for configuring NDB Cluster. For detailed descriptions and additional information about individual parameters, see Section 5.3.10, “NDB Cluster TCP/IP Connections”, or Section 5.3.12, “NDB Cluster Shared-Memory Connections”, as appropriate.

The following parameters apply to the config.ini file’s [computer] section:

The following parameters apply to the config.ini file’s [tcp] section:

The following parameters apply to the config.ini file’s [shm] section:

5.2.5 NDB Cluster mysqld Option and Variable Reference

The following table provides a list of the command-line options, server and status variables applicable within mysqld when it is running as an SQL node in an NDB Cluster. For a table showing all command-line options, server and status variables available for use with mysqld, see Server Option, System Variable, and Status Variable Reference.
5.3 NDB Cluster Configuration Files

Configuring NDB Cluster requires working with two files:

- **my.cnf**: Specifies options for all NDB Cluster executables. This file, with which you should be familiar from previous work with MySQL, must be accessible by each executable running in the cluster.

- **config.ini**: This file, sometimes known as the *global configuration file*, is read only by the NDB Cluster management server, which then distributes the information contained therein to all processes participating in the cluster. config.ini contains a description of each node involved in the cluster. This includes configuration parameters for data nodes and configuration parameters for connections between all nodes in the cluster. For a quick reference to the sections that can appear in this file, and what sorts of configuration parameters may be placed in each section, see Sections of the config.ini File.

Caching of configuration data.  
NDB uses *stateful configuration*. Rather than reading the global configuration file every time the management server is restarted, the management server caches the configuration the first time it is started, and thereafter, the global configuration file is read only when one of the following conditions is true:

- **The management server is started using the --initial option.**  
  When --initial is used, the global configuration file is re-read, any existing cache files are deleted, and the management server creates a new configuration cache.

- **The management server is started using the --reload option.**  
  The --reload option causes the management server to compare its cache with the global configuration file. If they differ, the management server creates a new configuration cache; any existing configuration cache is preserved, but not used. If the management server's cache and the global configuration file contain the same configuration data, then the existing cache is used, and no new cache is created.

- **The management server is started using --config-cache=FALSE.**  
  This disables --config-cache (enabled by default), and can be used to force the management server to bypass configuration caching altogether. In this case, the management server ignores any configuration files that may be present, always reading its configuration data from the config.ini file instead.

- **No configuration cache is found.**  
  In this case, the management server reads the global configuration file and creates a cache containing the same configuration data as found in the file.

Configuration cache files.  
The management server by default creates configuration cache files in a directory named *mysql-cluster* in the MySQL installation directory. (If you build NDB Cluster from source on a Unix system, the default location is /usr/local/mysql-cluster.) This can be overridden at runtime by starting the management server with the --configdir option. Configuration cache files are binary files named according to the pattern *ndb_node_id_config.bin.seq_id*, where *node_id* is the management server's node ID in the cluster, and *seq_id* is a cache identifier. Cache files are numbered sequentially using *seq_id*, in the order in which they are created. The management server uses the latest cache file as determined by the *seq_id*.

### Note

It is possible to roll back to a previous configuration by deleting later configuration cache files, or by renaming an earlier cache file so that it has a higher *seq_id*. However, since configuration cache files are written in a binary format, you should not attempt to edit their contents by hand.

For more information about the --configdir, --config-cache, --initial, and --reload options for the NDB Cluster management server, see Section 6.4, “ndb_mgmd — The NDB Cluster Management Server Daemon”.
We are continuously making improvements in NDB Cluster configuration and attempting to simplify this process. Although we strive to maintain backward compatibility, there may be times when introduce an incompatible change. In such cases we try to let NDB Cluster users know in advance if a change is not backward compatible. If you find such a change and we have not documented it, please report it in the MySQL bugs database using the instructions given in How to Report Bugs or Problems.

5.3.1 NDB Cluster Configuration: Basic Example

To support NDB Cluster, you should update `my.cnf` as shown in the following example. You may also specify these parameters on the command line when invoking the executables.

**Note**
The options shown here should not be confused with those that are used in `config.ini` global configuration files. Global configuration options are discussed later in this section.

```sql
# my.cnf
# example additions to my.cnf for NDB Cluster
# (valid in MySQL 8.0)
# enable ndbcluster storage engine, and provide connection string for
# management server host (default port is 1186)
[mysqld]
ndbcluster
ndb-connectstring=ndb_mgmd.mysql.com
# provide connection string for management server host (default port: 1186)
[ndbd]
connect-string=ndb_mgmd.mysql.com
# provide connection string for management server host (default port: 1186)
[ndb_mgm]
connect-string=ndb_mgmd.mysql.com
# provide location of cluster configuration file
[ndb_mgmd]
config-file=/etc/config.ini
(For more information on connection strings, see Section 5.3.3, “NDB Cluster Connection Strings”.)
```

```sql
# my.cnf
# example additions to my.cnf for NDB Cluster
# (works on all versions)
# enable ndbcluster storage engine, and provide connection string for management
# server host to the default port 1186
[mysqld]
ndbCluster
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

**Important**
Once you have started a `mysqld` process with the `NDBCLUSTER` and `ndb-connectstring` parameters in the `[mysqld]` in the `my.cnf` file as shown previously, you cannot execute any `CREATE TABLE` or `ALTER TABLE` statements without having actually started the cluster. Otherwise, these statements fail with an error. *This is by design.*

You may also use a separate `[mysql_cluster]` section in the cluster `my.cnf` file for settings to be read and used by all executables:

```sql
# cluster-specific settings
[mysql_cluster]
ndb-connectstring=ndb_mgmd.mysql.com:1186
```

For additional NDB variables that can be set in the `my.cnf` file, see Section 5.3.9.2, “NDB Cluster System Variables”.

The NDB Cluster global configuration file is by convention named `config.ini` (but this is not required). If needed, it is read by `ndb_mgmd` at startup and can be placed in any location that
can be read by it. The location and name of the configuration are specified using `--config-file=\path_name` with `ndb_mgmd` on the command line. This option has no default value, and is ignored if `ndb_mgmd` uses the configuration cache.

The global configuration file for NDB Cluster uses INI format, which consists of sections preceded by section headings (surrounded by square brackets), followed by the appropriate parameter names and values. One deviation from the standard INI format is that the parameter name and value can be separated by a colon (:) as well as the equal sign (=); however, the equal sign is preferred. Another deviation is that sections are not uniquely identified by section name. Instead, unique sections (such as two different nodes of the same type) are identified by a unique ID specified as a parameter within the section.

Default values are defined for most parameters, and can also be specified in `config.ini`. To create a default value section, simply add the word `default` to the section name. For example, an `[ndbd]` section contains parameters that apply to a particular data node, whereas an `[ndbd default]` section contains parameters that apply to all data nodes. Suppose that all data nodes should use the same data memory size. To configure them all, create an `ndbd default` section that contains a `DataMemory` line to specify the data memory size.

If used, the `ndbd default` section must precede any `ndbd` sections in the configuration file. This is also true for `default` sections of any other type.

**Note**

In some older releases of NDB Cluster, there was no default value for `NoOfReplicas`, which always had to be specified explicitly in the `ndbd default` section. Although this parameter now has a default value of 2, which is the recommended setting in most common usage scenarios, it is still recommended practice to set this parameter explicitly.

The global configuration file must define the computers and nodes involved in the cluster and on which computers these nodes are located. An example of a simple configuration file for a cluster consisting of one management server, two data nodes and two MySQL servers is shown here:

```ini
# file "config.ini" - 2 data nodes and 2 SQL nodes
# This file is placed in the startup directory of ndb_mgmd (the
# management server)
# The first MySQL Server can be started from any host. The second
# can be started only on the host mysqld_5.mysql.com
[ndbd default]
NoOfReplicas= 2
DataDir= /var/lib/mysql-cluster
[ndb_mgmd]
Hostname= ndb_mgmd.mysql.com
DataDir= /var/lib/mysql-cluster
[ndbd]
HostName= ndbd_2.mysql.com
[ndbd]
HostName= ndbd_3.mysql.com
[mysqld]
[mysqld]
HostName= mysqld_5.mysql.com
```

**Note**

The preceding example is intended as a minimal starting configuration for purposes of familiarization with NDB Cluster, and is almost certain not to be sufficient for production settings. See Section 5.3.2, "Recommended Starting Configuration for NDB Cluster", which provides a more complete example starting configuration.

Each node has its own section in the `config.ini` file. For example, this cluster has two data nodes, so the preceding configuration file contains two `ndbd` sections defining these nodes.
Recommended Starting Configuration for NDB Cluster

Note

Do not place comments on the same line as a section heading in the config.ini file; this causes the management server not to start because it cannot parse the configuration file in such cases.

Sections of the config.ini File

There are six different sections that you can use in the config.ini configuration file, as described in the following list:

- **[computer]**: Defines cluster hosts. This is not required to configure a viable NDB Cluster, but may be used as a convenience when setting up a large cluster. See Section 5.3.4, “Defining Computers in an NDB Cluster”, for more information.

- **[ndbd]**: Defines a cluster data node (ndbd process). See Section 5.3.6, “Defining NDB Cluster Data Nodes”, for details.

- **[mysqld]**: Defines the cluster's MySQL server nodes (also called SQL or API nodes). For a discussion of SQL node configuration, see Section 5.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- **[mgm]** or **[ndb_mgmd]**: Defines a cluster management server (MGM) node. For information concerning the configuration of management nodes, see Section 5.3.5, “Defining an NDB Cluster Management Server”.

- **[tcp]**: Defines a TCP/IP connection between cluster nodes, with TCP/IP being the default transport protocol. Normally, [tcp] or [tcp default] sections are not required to set up an NDB Cluster, as the cluster handles this automatically; however, it may be necessary in some situations to override the defaults provided by the cluster. See Section 5.3.10, “NDB Cluster TCP/IP Connections”, for information about available TCP/IP configuration parameters and how to use them. (You may also find Section 5.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections” to be of interest in some cases.)

- **[shm]**: Defines shared-memory connections between nodes. In MySQL 8.0, it is enabled by default, but should still be considered experimental. For a discussion of SHM interconnects, see Section 5.3.12, “NDB Cluster Shared-Memory Connections”.

- **[sci]**: Defines Scalable Coherent Interface connections between cluster data nodes. Not supported in NDB 8.0.

You can define default values for each section. If used, a default section should come before any other sections of that type. For example, an [ndbd default] section should appear in the configuration file before any [ndbd] sections.

NDB Cluster parameter names are case-insensitive, unless specified in MySQL Server my.cnf or my.ini files.

5.3.2 Recommended Starting Configuration for NDB Cluster

Achieving the best performance from an NDB Cluster depends on a number of factors including the following:

- NDB Cluster software version
- Numbers of data nodes and SQL nodes
- Hardware
- Operating system
- Amount of data to be stored
• Size and type of load under which the cluster is to operate

Therefore, obtaining an optimum configuration is likely to be an iterative process, the outcome of which can vary widely with the specifics of each NDB Cluster deployment. Changes in configuration are also likely to be indicated when changes are made in the platform on which the cluster is run, or in applications that use the NDB Cluster’s data. For these reasons, it is not possible to offer a single configuration that is ideal for all usage scenarios. However, in this section, we provide a recommended base configuration.

**Starting config.ini file.** The following `config.ini` file is a recommended starting point for configuring a cluster running NDB Cluster 8.0:

```ini
# TCP PARAMETERS
[tcp default]
SendBufferMemory=2M
ReceiveBufferMemory=2M

# Increasing the sizes of these 2 buffers beyond the default values helps prevent bottlenecks due to slow disk I/O.

# MANAGEMENT NODE PARAMETERS
[ndb_mgmd default]
DataDir=path/to/management/server/data/directory

# It is possible to use a different data directory for each management server, but for ease of administration it is preferable to be consistent.
[ndb_mgmd]
HostName=management-server-A-hostname
NodeId=management-server-A-nodeid
[ndb_mgmd]
HostName=management-server-B-hostname
NodeId=management-server-B-nodeid

# Using 2 management servers helps guarantee that there is always an arbitrator in the event of network partitioning, and so is recommended for high availability. Each management server must be identified by a HostName. You may for the sake of convenience specify a NodeId for any management server, although one is allocated for it automatically; if you do so, it must be in the range 1-255 inclusive and must be unique among all IDs specified for cluster nodes.

# DATA NODE PARAMETERS
[ndbd default]
NoOfReplicas=2

# Using two fragment replicas is recommended to guarantee availability of data; using only one fragment replica does not provide any redundancy, which means that the failure of a single data node causes the entire cluster to shut down. As of NDB 8.0.19, it is also possible (but not required) to use more than two fragment replicas, although two fragment replicas are sufficient to provide high availability.
LockPagesInMainMemory=1

# On Linux and Solaris systems, setting this parameter locks data node processes into memory. Doing so prevents them from swapping to disk, which can severely degrade cluster performance.
DataMemory=3456M

# The value provided for DataMemory assumes 4 GB RAM per data node. However, for best results, you should first calculate the memory that would be used based on the data you actually plan to store (you may find the `ndb_size.pl` utility helpful in estimating this), then allow an extra 20% over the calculated values. Naturally, you should ensure that each data node host has at least as much physical memory as the sum of these two values.

ODirect=1

# Enabling this parameter causes NDBCLUSTER to try using O_DIRECT for local checkpoints and redo logs; this can reduce load on CPUs. We recommend doing so when using NDB Cluster on systems running Linux kernel 2.6 or later.
NoOfFragmentLogFiles=300
DataDir=path/to/data/node/data/directory
MaxNoOfConcurrentOperations=100000
SchedulerSpinTimer=400
SchedulerExecutionTimer=100
RealTimeScheduler=1

# Setting these parameters allows you to take advantage of real-time scheduling
```

114
Recommended Starting Configuration for NDB Cluster

- # of NDB threads to achieve increased throughput when using ndbd. They are not needed when using ndbmtd; in particular, you should not set RealTimeScheduler for ndbmtd data nodes.
  - TimeBetweenGlobalCheckpoints=1000
  - TimeBetweenEpochs=200
  - RedoBuffer=32M
  - CompressedLCP=1
  - CompressedBackup=1

- Enabling CompressedLCP and CompressedBackup causes, respectively, local checkpoint files and backup files to be compressed, which can result in a space savings of up to 50% over noncompressed LCPs and backups.
- MaxNoOfLocalScans=64
- MaxNoOfTables=1024
- MaxNoOfOrderedIndexes=256

[ndbd]
HostName=data-node-A-hostname

[ndbd]
HostName=data-node-B-hostname

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0

# On systems with multiple CPUs, these parameters can be used to lock NDBCLUSTER threads to specific CPUs

SQL NODE / API NODE PARAMETERS

[mysqld]
HostName=sql-node-A-hostname

[mysqld]
HostName=sql-node-B-hostname

LockExecuteThreadToCPU=1
LockMaintThreadsToCPU=0

# You must have an [ndbd] section for every data node in the cluster;
# each of these sections must include a HostName. Each section may
# optionally include a NodeId for convenience, but in most cases, it is
# sufficient to allow the cluster to allocate node IDs dynamically. If
# you do specify the node ID for a data node, it must be in the range 1
to 144 inclusive and must be unique among all IDs specified for
# cluster nodes. (Previous to NDB 8.0.18, this range was 1 to 48 inclusive.)
# SQL NODE / API NODE PARAMETERS

Recommended my.cnf options for SQL nodes. MySQL Servers acting as NDB Cluster SQL nodes must always be started with the --ndbcluster and --ndb-connectstring options, either on the command line or in my.cnf. In addition, set the following options for all mysql processes in the cluster, unless your setup requires otherwise:

- --ndb-use-exact-count=0
- --ndb-index-stat-enable=0
- --ndb-force-send=1
- --optimizer-switch=engine_condition_pushdown=on
5.3.3 NDB Cluster Connection Strings

With the exception of the NDB Cluster management server (ndb_mgmd), each node that is part of an NDB Cluster requires a connection string that points to the management server's location. This connection string is used in establishing a connection to the management server as well as in performing other tasks depending on the node's role in the cluster. The syntax for a connection string is as follows:

```
[nodeid=node_id, ]host-definition[, host-definition[, ...]]
host-definition:
  host_name[:port_number]
```

- **node_id** is an integer greater than or equal to 1 which identifies a node in `config.ini`. **host_name** is a string representing a valid Internet host name or IP address. **port_number** is an integer referring to a TCP/IP port number.

  **example 1 (long):** "nodeid=2,myhost1:1100,myhost2:1100,198.51.100.3:1200"

  **example 2 (short):** "myhost1"

- **localhost:1186** is used as the default connection string value if none is provided. If **port_num** is omitted from the connection string, the default port is 1186. This port should always be available on the network because it has been assigned by IANA for this purpose (see [http://www.iana.org/assignments/port-numbers](http://www.iana.org/assignments/port-numbers) for details).

By listing multiple host definitions, it is possible to designate several redundant management servers. An NDB Cluster data or API node attempts to contact successive management servers on each host in the order specified, until a successful connection has been established.

It is also possible to specify in a connection string one or more bind addresses to be used by nodes having multiple network interfaces for connecting to management servers. A bind address consists of a hostname or network address and an optional port number. This enhanced syntax for connection strings is shown here:

```
[nodeid=node_id, ]
[bind-address=host-definition, ]
host-definition[; bind-address=host-definition]
host-definition[; bind-address=host-definition]
[, ...]]
host-definition:
  host_name[;port_number]
```

If a single bind address is used in the connection string prior to specifying any management hosts, then this address is used as the default for connecting to any of them (unless overridden for a given management server; see later in this section for an example). For example, the following connection string causes the node to use **198.51.100.242** regardless of the management server to which it connects:

```
bind-address=198.51.100.242, poseidon:1186, perch:1186
```

If a bind address is specified following a management host definition, then it is used only for connecting to that management node. Consider the following connection string:

```
poseidon:1186;bind-address=localhost, perch:1186;bind-address=198.51.100.242
```

In this case, the node uses **localhost** to connect to the management server running on the host named **poseidon** and **198.51.100.242** to connect to the management server running on the host named **perch**.

You can specify a default bind address and then override this default for one or more specific management hosts. In the following example, **localhost** is used for connecting to the management server running on host **poseidon**: since **198.51.100.242** is specified first (before any management server definitions), it is the default bind address and so is used for connecting to the management servers on hosts **perch** and **orca**.
There are a number of different ways to specify the connection string:

- Each executable has its own command-line option which enables specifying the management server at startup. (See the documentation for the respective executable.)

- It is also possible to set the connection string for all nodes in the cluster at once by placing it in a `[mysql_cluster]` section in the management server’s `my.cnf` file.

- For backward compatibility, two other options are available, using the same syntax:
  1. Set the `NDB_CONNECTSTRING` environment variable to contain the connection string.
  2. Write the connection string for each executable into a text file named `Ndb.cfg` and place this file in the executable’s startup directory.

However, these are now deprecated and should not be used for new installations.

The recommended method for specifying the connection string is to set it on the command line or in the `my.cnf` file for each executable.

### 5.3.4 Defining Computers in an NDB Cluster

The `[computer]` section has no real significance other than serving as a way to avoid the need of defining host names for each node in the system. All parameters mentioned here are required.

- **Id**

  This is a unique identifier, used to refer to the host computer elsewhere in the configuration file.

  **Important**
  
  The computer ID is **not** the same as the node ID used for a management, API, or data node. Unlike the case with node IDs, you cannot use `NodeId` in place of `Id` in the `[computer]` section of the `config.ini` file.

- **HostName**

  This is the computer’s hostname or IP address.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the <code>--initial</code> option</td>
</tr>
</tbody>
</table>

### 5.3.5 Defining an NDB Cluster Management Server

The `[ndb_mgmd]` section is used to configure the behavior of the management server. If multiple management servers are employed, you can specify parameters common to all of them in an
[ndb_mgmd default] section. [mgm] and [mgm default] are older aliases for these, supported for backward compatibility.

All parameters in the following list are optional and assume their default values if omitted.

- **Id**

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255, inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

  **Note**

  Data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 144. (In NDB 8.0.17 and earlier, the maximum value for a data node ID was 48.)

The use of the **Id** parameter for identifying management nodes is deprecated in favor of **NodeId**. Although **Id** continues to be supported for backward compatibility, it now generates a warning and is subject to removal in a future version of NDB Cluster.

- **NodeId**

Each node in the cluster has a unique identity. For a management node, this is represented by an integer value in the range 1 to 255 inclusive. This ID is used by all internal cluster messages for addressing the node, and so must be unique for each NDB Cluster node, regardless of the type of node.

  **Note**

  As of NDB 8.0.18, data node IDs must be less than 145. (Previously, this was less than 49.) If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for management nodes (and API nodes) to values greater than 144.

**NodeId** is the preferred parameter name to use when identifying management nodes. Although the older **Id** continues to be supported for backward compatibility, it is now deprecated and generates a warning when used; it is also subject to removal in a future NDB Cluster release.

- **ExecuteOnComputer**

This refers to the **Id** set for one of the computers defined in a [computer] section of the **config.ini** file.

  **Important**

  This parameter is deprecated, and is subject to removal in a future release. Use the **HostName** parameter instead.

- **PortNumber**
Defining an NDB Cluster Management Server

This is the port number on which the management server listens for configuration requests and management commands.

- The node ID for this node can be given out only to connections that explicitly request it. A management server that requests "any" node ID cannot use this one. This parameter can be used when running multiple management servers on the same host, and HostName is not sufficient for distinguishing among processes. Intended for use in testing.

- **HostName**

  Specifying this parameter defines the hostname of the computer on which the management node is to reside. To specify a hostname other than localhost, either this parameter or ExecuteOnComputer is required.

- **LocationDomainId**

  Assigns a management node to a specific availability domain (also known as an availability zone) within a cloud. By informing NDB which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:
  - If requested data is not found on the same node, reads can be directed to another node in the same availability domain.
  - Communication between nodes in different availability domains are guaranteed to use NDB transporters' WAN support without any further manual intervention.
  - The transporter's group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.
  - The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.

  LocationDomainId takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

- **LogDestination**

  This parameter specifies where to send cluster logging information. There are three options in this regard—CONSOLE, SYSLOG, and FILE—with FILE being the default:

  - **CONSOLE** outputs the log to stdout:

  ```
  CONSOLE
  ```

  - **SYSLOG** sends the log to a syslog facility, possible values being one of auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0, local1, local2, local3, local4, local5, local6, or local7.

  ```
  SYSLOG:facility=syslog
  ```

  **Note**

  Not every facility is necessarily supported by every operating system.

  ```
  SYSLOG:facility=syslog
  ```

  - **FILE** pipes the cluster log output to a regular file on the same machine. The following values can be specified:
• **filename**: The name of the log file.

The default log file name used in such cases is `ndb_nodeid_cluster.log`.

• **maxsize**: The maximum size (in bytes) to which the file can grow before logging rolls over to a new file. When this occurs, the old log file is renamed by appending \( .N \) to the file name, where \( N \) is the next number not yet used with this name.

• **maxfiles**: The maximum number of log files.

```
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
```

The default value for the `FILE` parameter is

```
FILE:filename=ndb_node_id_cluster.log,maxsize=1000000,maxfiles=6,where
node_id is the ID of the node.
```

It is possible to specify multiple log destinations separated by semicolons as shown here:

```
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgmd
```

• **ArbitrationRank**

This parameter is used to define which nodes can act as arbitrators. Only management nodes and SQL nodes can be arbitrators. `ArbitrationRank` can take one of the following values:

• **0**: The node is never used as an arbitrator.

• **1**: The node has high priority; that is, it is preferred as an arbitrator over low-priority nodes.

• **2**: Indicates a low-priority node which is used as an arbitrator only if a node with a higher priority is not available for that purpose.

Normally, the management server should be configured as an arbitrator by setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

You can disable arbitration completely either by setting `ArbitrationRank` to 0 on all management and SQL nodes, or by setting the `Arbitration` parameter in the `[ndbd default]` section of the `config.ini` global configuration file. Setting `Arbitration` causes any settings for `ArbitrationRank` to be disregarded.

• **ArbitrationDelay**

An integer value which causes the management server's responses to arbitration requests to be delayed by that number of milliseconds. By default, this value is 0; it is normally not necessary to change it.

• **DataDir**

This specifies the directory where output files from the management server are placed. These files include cluster log files, process output files, and the daemon's process ID (PID) file. (For log files, this location can be overridden by setting the `FILE` parameter for `LogDestination`, as discussed previously in this section.)

The default value for this parameter is the directory in which `ndb_mgmd` is located.
• **PortNumberStats**

This parameter specifies the port number used to obtain statistical information from an NDB Cluster management server. It has no default value.

• **Wan**

Use WAN TCP setting as default.

• **HeartbeatThreadPriority**

Set the scheduling policy and priority of heartbeat threads for management and API nodes.

The syntax for setting this parameter is shown here:

```
HeartbeatThreadPriority = policy[, priority]
policy: {FIFO | RR}
```

When setting this parameter, you must specify a policy. This is one of **FIFO** (first in, first out) or **RR** (round robin). The policy value is followed optionally by the priority (an integer).

• **ExtraSendBufferMemory**

This parameter specifies the amount of transporter send buffer memory to allocate in addition to any that has been set using **TotalSendBufferMemory**, **SendBufferMemory**, or both.

• **TotalSendBufferMemory**

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

• **HeartbeatIntervalMgmdMgmd**

Specify the interval between heartbeat messages used to determine whether another management node is on contact with this one. The management node waits after 3 of these intervals to declare the connection dead; thus, the default setting of 1500 milliseconds causes the management node to wait for approximately 1600 ms before timing out.

**Note**

After making changes in a management node's configuration, it is necessary to perform a rolling restart of the cluster for the new configuration to take effect.

To add new management servers to a running NDB Cluster, it is also necessary to perform a rolling restart of all cluster nodes after modifying any existing *config.ini* files. For more information about issues arising when using multiple management nodes, see Section 3.7.10, “Limitations Relating to Multiple NDB Cluster Nodes”.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:
Table 5.2 NDB Cluster restart types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the <code>--initial</code> option</td>
</tr>
</tbody>
</table>

5.3.6 Defining NDB Cluster Data Nodes

The `[ndbd]` and `[ndbd default]` sections are used to configure the behavior of the cluster's data nodes.

`[ndbd]` and `[ndbd default]` are always used as the section names whether you are using `ndbd` or `ndbmtd` binaries for the data node processes.

There are many parameters which control buffer sizes, pool sizes, timeouts, and so forth. The only mandatory parameter is either one of `ExecuteOnComputer` or `HostName`; this must be defined in the local `[ndbd]` section.

The parameter `NoOfReplicas` should be defined in the `[ndbd default]` section, as it is common to all Cluster data nodes. It is not strictly necessary to set `NoOfReplicas`, but it is good practice to set it explicitly.

Most data node parameters are set in the `[ndbd default]` section. Only those parameters explicitly stated as being able to set local values are permitted to be changed in the `[ndbd]` section. Where present, `HostName`, `NodeId` and `ExecuteOnComputer` must be defined in the local `[ndbd]` section, and not in any other section of `config.ini`. In other words, settings for these parameters are specific to one data node.

For those parameters affecting memory usage or buffer sizes, it is possible to use `K`, `M`, or `G` as a suffix to indicate units of 1024, 1024×1024, or 1024×1024×1024. (For example, `100K` means 100 × 1024 = 102400.)

Parameter names and values are case-insensitive, unless used in a MySQL Server `my.cnf` or `my.ini` file, in which case they are case-sensitive.

Information about configuration parameters specific to NDB Cluster Disk Data tables can be found later in this section (see Disk Data Configuration Parameters).

All of these parameters also apply to `ndbmtd` (the multithreaded version of `ndbd`). Three additional data node configuration parameters—`MaxNoOfExecutionThreads`, `ThreadConfig`, and `NoOfFragmentLogParts`—apply to `ndbmtd` only; these have no effect when used with `ndbd`. For more information, see Multi-Threading Configuration Parameters (ndbmtd). See also Section 6.3, "ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)".

Identifying data nodes. The `NodeId` or `Id` value (that is, the data node identifier) can be allocated on the command line when the node is started or in the configuration file.

- `NodeId` 

A unique node ID is used as the node’s address for all cluster internal messages. For data nodes, this is an integer in the range 1 to 144 inclusive. (In NDB 8.0.17 and earlier, this was 1 to 48 inclusive.) Each node in the cluster must have a unique identifier.

`NodeId` is the only supported parameter name to use when identifying data nodes.
• **ExecuteOnComputer**

This refers to the `Id` set for one of the computers defined in a `[computer]` section.

```
Important
This parameter is deprecated, and is subject to removal in a future release.
Use the HostName parameter instead.
```

•

The node ID for this node can be given out only to connections that explicitly request it. A management server that requests “any” node ID cannot use this one. This parameter can be used when running multiple management servers on the same host, and HostName is not sufficient for distinguishing among processes. Intended for use in testing.

• **HostName**

Specifying this parameter defines the hostname of the computer on which the data node is to reside. To specify a hostname other than `localhost`, either this parameter or ExecuteOnComputer is required.

• **ServerPort**

Each node in the cluster uses a port to connect to other nodes. By default, this port is allocated dynamically in such a way as to ensure that no two nodes on the same host computer receive the same port number, so it should normally not be necessary to specify a value for this parameter.

However, if you need to be able to open specific ports in a firewall to permit communication between data nodes and API nodes (including SQL nodes), you can set this parameter to the number of the desired port in an `[ndbd]` section or (if you need to do this for multiple data nodes) the `[ndbd default]` section of the `config.ini` file, and then open the port having that number for incoming connections from SQL nodes, API nodes, or both.

```
Note
Connections from data nodes to management nodes is done using the ndb_mgmd management port (the management server’s PortNumber) so outgoing connections to that port from any data nodes should always be permitted.
```

• **TcpBind_INADDR_ANY**

Setting this parameter to `TRUE` or `1` binds `IP_ADDR_ANY` so that connections can be made from anywhere (for autogenerated connections). The default is `FALSE` (0).

• **NodeGroup**

This parameter can be used to assign a data node to a specific node group. It is read only when the cluster is started for the first time, and cannot be used to reassign a data node to a different node group online. It is generally not desirable to use this parameter in the `[ndbd default]` section of the `config.ini` file, and care must be taken not to assign nodes to node groups in such a way that an invalid numbers of nodes are assigned to any node groups.

The **NodeGroup** parameter is chiefly intended for use in adding a new node group to a running NDB Cluster without having to perform a rolling restart. For this purpose, you should set it to 65536 (the maximum value). You are not required to set a **NodeGroup** value for all cluster data nodes, only for those nodes which are to be started and added to the cluster as a new node group at a later
Defining NDB Cluster Data Nodes

time. For more information, see Section 7.7.3, “Adding NDB Cluster Data Nodes Online: Detailed Example”.

• LocationDomainId

Assigns a data node to a specific availability domain (also known as an availability zone) within a cloud. By informing NDB which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:

• If requested data is not found on the same node, reads can be directed to another node in the same availability domain.

• Communication between nodes in different availability domains are guaranteed to use NDB transporters’ WAN support without any further manual intervention.

• The transporter's group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.

• The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.

LocationDomainId takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

• NoOfReplicas

This global parameter can be set only in the [ndbd default] section, and defines the number of fragment replicas for each table stored in the cluster. This parameter also specifies the size of node groups. A node group is a set of nodes all storing the same information.

Node groups are formed implicitly. The first node group is formed by the set of data nodes with the lowest node IDs, the next node group by the set of the next lowest node identities, and so on. By way of example, assume that we have 4 data nodes and that NoOfReplicas is set to 2. The four data nodes have node IDs 2, 3, 4 and 5. Then the first node group is formed from nodes 2 and 3, and the second node group by nodes 4 and 5. It is important to configure the cluster in such a manner that nodes in the same node groups are not placed on the same computer because a single hardware failure would cause the entire cluster to fail.

If no node IDs are provided, the order of the data nodes is the determining factor for the node group. Whether or not explicit assignments are made, they can be viewed in the output of the management client’s SHOW command.

The default value for NoOfReplicas is 2. This is the recommended value for most production environments, and prior to NDB 8.0.18, it was the maximum value supported. Beginning with NDB 8.0.19, setting this parameter’s value to 3 or 4 is fully tested and supported in production.

Warning

Setting NoOfReplicas to 1 means that there is only a single copy of all Cluster data; in this case, the loss of a single data node causes the cluster to fail because there are no additional copies of the data stored by that node.

The number of data nodes in the cluster must be evenly divisible by the value of this parameter. For example, if there are two data nodes, then NoOfReplicas must be equal to either 1 or 2, since 2/3 and 2/4 both yield fractional values; if there are four data nodes, then NoOfReplicas must be equal to 1, 2, or 4.
Defining NDB Cluster Data Nodes

- **DataDir**

  This parameter specifies the directory where trace files, log files, pid files and error logs are placed. The default is the data node process working directory.

- **FileSystemPath**

  This parameter specifies the directory where all files created for metadata, REDO logs, UNDO logs (for Disk Data tables), and data files are placed. The default is the directory specified by `DataDir`. **Note**

  This directory must exist before the `ndbd` process is initiated.

  The recommended directory hierarchy for NDB Cluster includes `/var/lib/mysql-cluster`, under which a directory for the node's file system is created. The name of this subdirectory contains the node ID. For example, if the node ID is 2, this subdirectory is named `ndb_2_fs`.

- **BackupDataDir**

  This parameter specifies the directory in which backups are placed. **Important**

  The string '/BACKUP' is always appended to this value. For example, if you set the value of `BackupDataDir` to `/var/lib/cluster-data`, then all backups are stored under `/var/lib/cluster-data/BACKUP`. This also means that the effective default backup location is the directory named BACKUP under the location specified by the `FileSystemPath` parameter.

### Data Memory, Index Memory, and String Memory

`DataMemory` and `IndexMemory` are `[ndbd]` parameters specifying the size of memory segments used to store the actual records and their indexes. In setting values for these, it is important to understand how `DataMemory` is used, as it usually needs to be updated to reflect actual usage by the cluster.

**Note**

`IndexMemory` is deprecated, and subject to removal in a future version of NDB Cluster. See the descriptions that follow for further information.

- **DataMemory**

  This parameter defines the amount of space (in bytes) available for storing database records. The entire amount specified by this value is allocated in memory, so it is extremely important that the machine has sufficient physical memory to accommodate it.

  The memory allocated by `DataMemory` is used to store both the actual records and indexes. There is a 16-byte overhead on each record; an additional amount for each record is incurred because it is stored in a 32KB page with 128 byte page overhead (see below). There is also a small amount wasted per page due to the fact that each record is stored in only one page.

  For variable-size table attributes, the data is stored on separate data pages, allocated from `DataMemory`. Variable-length records use a fixed-size part with an extra overhead of 4 bytes to reference the variable-size part. The variable-size part has 2 bytes overhead plus 2 bytes per attribute.
As of NDB 8.0.18, the maximum record size is 30000 bytes. Previously, this was 14000 bytes.

Resources assigned to `DataMemory` are used for storing all data and indexes. (Any memory configured as `IndexMemory` is automatically added to that used by `DataMemory` to form a common resource pool.)

Currently, NDB Cluster can use a maximum of 512 MB for hash indexes per partition, which means in some cases it is possible to get `Table is full` errors in MySQL client applications even when `ndb_mgm -e "ALL REPORT MEMORYUSAGE"` shows significant free `DataMemory`. This can also pose a problem with data node restarts on nodes that are heavily loaded with data.

You can control the number of partitions per local data manager for a given table by setting the `NDB_TABLE` option `PARTITION_BALANCE` to one of the values `FOR_RA_BY_LDM`, `FOR_RA_BY_LDM_X_2`, `FOR_RA_BY_LDM_X_3`, or `FOR_RA_BY_LDM_X_4`, for 1, 2, 3, or 4 partitions per LDM, respectively, when creating the table (see Setting `NDB_TABLE` Options).

**Note**

In previous versions of NDB Cluster it was possible to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the `MAX_ROWS` option for `CREATE TABLE`. While still supported for backward compatibility, using `MAX_ROWS` for this purpose is deprecated; you should use `PARTITION_BALANCE` instead.

You can also use the `MinFreePct` configuration parameter to help avoid problems with node restarts.

The memory space allocated by `DataMemory` consists of 32KB pages, which are allocated to table fragments. Each table is normally partitioned into the same number of fragments as there are data nodes in the cluster. Thus, for each node, there are the same number of fragments as are set in `NoOfReplicas`.

Once a page has been allocated, it is currently not possible to return it to the pool of free pages, except by deleting the table. (This also means that `DataMemory` pages, once allocated to a given table, cannot be used by other tables.) Performing a data node recovery also compresses the partition because all records are inserted into empty partitions from other live nodes.

The `DataMemory` memory space also contains UNDO information: For each update, a copy of the unaltered record is allocated in the `DataMemory`. There is also a reference to each copy in the ordered table indexes. Unique hash indexes are updated only when the unique index columns are updated, in which case a new entry in the index table is inserted and the old entry is deleted upon commit. For this reason, it is also necessary to allocate enough memory to handle the largest transactions performed by applications using the cluster. In any case, performing a few large transactions holds no advantage over using many smaller ones, for the following reasons:

- Large transactions are not any faster than smaller ones
- Large transactions increase the number of operations that are lost and must be repeated in event of transaction failure
- Large transactions use more memory

The default value for `DataMemory` in NDB 8.0 is 98MB. The minimum value is 1MB. There is no maximum size, but in reality the maximum size has to be adapted so that the process does not start swapping when the limit is reached. This limit is determined by the amount of physical RAM available on the machine and by the amount of memory that the operating system may commit to any one process. 32-bit operating systems are generally limited to 2–4GB per process; 64-bit operating systems can use more. For large databases, it may be preferable to use a 64-bit operating system for this reason.
Defining NDB Cluster Data Nodes

• IndexMemory

The IndexMemory parameter is deprecated (and subject to future removal); any memory assigned to IndexMemory is allocated instead to the same pool as DataMemory, which is solely responsible for all resources needed for storing data and indexes in memory. In NDB 8.0, the use of IndexMemory in the cluster configuration file triggers a warning from the management server.

You can estimate the size of a hash index using this formula:

\[
\text{size} = (\text{fragments} \times 32\text{K}) + (\text{rows} \times 18) \times \text{fragment_replicas}
\]

fragments is the number of fragments, fragment_replicas is the number of fragment replicas (normally 2), and rows is the number of rows. If a table has one million rows, eight fragments, and two fragment replicas, the expected index memory usage is calculated as shown here:

\[
\begin{align*}
((8 \times 32\text{K}) + (1000000 \times 18)) \times 2 &= ((8 \times 32768) + (1000000 \times 18)) \times 2 \\
&= (262144 + 18000000) \times 2 \\
&= 18262144 \times 2 = 36524288 \text{ bytes} = -35\text{MB}
\end{align*}
\]

Index statistics for ordered indexes (when these are enabled) are stored in the mysql.ndb_index_stat_sample table. Since this table has a hash index, this adds to index memory usage. An upper bound to the number of rows for a given ordered index can be calculated as follows:

\[
\begin{align*}
\text{sample_size} &= \text{key_size} + ((\text{key_attributes} + 1) \times 4) \\
\text{sample_rows} &= \frac{\text{IndexStatSaveSize} \times ((0.01 \times \text{IndexStatSaveScale} \times \log_2(\text{rows} \times \text{sample_size})) + 1)}{\text{sample_size}}
\end{align*}
\]

In the preceding formula, key_size is the size of the ordered index key in bytes, key_attributes is the number of attributes in the ordered index key, and rows is the number of rows in the base table.

Assume that table t1 has 1 million rows and an ordered index named ix1 on two four-byte integers. Assume in addition that IndexStatSaveSize and IndexStatSaveScale are set to their default values (32K and 100, respectively). Using the previous 2 formulas, we can calculate as follows:

\[
\begin{align*}
\text{sample_size} &= 8 + ((1 + 2) \times 4) = 20 \text{ bytes} \\
\text{sample_rows} &= 32\text{K} \\
&= 32768 \times ((0.01 \times 100 \times \log_2(1000000 \times 20)) + 1) \\
&= 32768 \times (1 \times -16.811) + 1) / 20 \\
&= 32768 \times -16.811 / 20 \\
&= -29182 \text{ rows}
\end{align*}
\]

The expected index memory usage is thus 2 \times 18 \times -29182 = -1050550 bytes.

In NDB 8.0, the minimum and default value for this parameter is 0 (zero).

• StringMemory

This parameter determines how much memory is allocated for strings such as table names, and is specified in an [ndbd] or [ndbd default] section of the config.ini file. A value between 0 and 100 inclusive is interpreted as a percent of the maximum default value, which is calculated based on a number of factors including the number of tables, maximum table name size, maximum
size of .FRM files, MaxNoOfTriggers, maximum column name size, and maximum default column value.

A value greater than 100 is interpreted as a number of bytes.

The default value is 25—that is, 25 percent of the default maximum.

Under most circumstances, the default value should be sufficient, but when you have a great many NDB tables (1000 or more), it is possible to get Error 773 Out of string memory, please modify StringMemory config parameter: Permanent error: Schema error, in which case you should increase this value. 25 (25 percent) is not excessive, and should prevent this error from recurring in all but the most extreme conditions.

The following example illustrates how memory is used for a table. Consider this table definition:

```sql
CREATE TABLE example (  
a INT NOT NULL,  
b INT NOT NULL,  
c INT NOT NULL,  
PRIMARY KEY(a),  
UNIQUE(b)  
) ENGINE=NDBCLUSTER;
```

For each record, there are 12 bytes of data plus 12 bytes overhead. In addition, we have two ordered indexes on columns a and b consuming roughly 10 bytes each per record. There is a primary key hash index on the base table using roughly 29 bytes per record. The unique constraint is implemented by a separate table with b as primary key and a as a column. This other table consumes an additional 29 bytes of index memory per record in the example table as well 8 bytes of record data plus 12 bytes of overhead.

Thus, for one million records, we need 58MB for index memory to handle the hash indexes for the primary key and the unique constraint. We also need 64MB for the records of the base table and the unique index table, plus the two ordered index tables.

You can see that hash indexes takes up a fair amount of memory space; however, they provide very fast access to the data in return. They are also used in NDB Cluster to handle uniqueness constraints.

Currently, the only partitioning algorithm is hashing and ordered indexes are local to each node. Thus, ordered indexes cannot be used to handle uniqueness constraints in the general case.

An important point for both IndexMemory and DataMemory is that the total database size is the sum of all data memory and all index memory for each node group. Each node group is used to store replicated information, so if there are four nodes with two fragment replicas, there are two node groups. Thus, the total data memory available is 2 × DataMemory for each data node.

It is highly recommended that DataMemory and IndexMemory be set to the same values for all nodes. Data distribution is even over all nodes in the cluster, so the maximum amount of space available for any node can be no greater than that of the smallest node in the cluster.

DataMemory can be changed, but decreasing it can be risky; doing so can easily lead to a node or even an entire NDB Cluster that is unable to restart due to there being insufficient memory space. Increasing these values should be acceptable, but it is recommended that such upgrades are performed in the same manner as a software upgrade, beginning with an update of the configuration file, and then restarting the management server followed by restarting each data node in turn.

MinFreePct. A proportion (5% by default) of data node resources including DataMemory is kept in reserve to insure that the data node does not exhaust its memory when performing a restart. This can be adjusted using the MinFreePct data node configuration parameter (default 5).

Updates do not increase the amount of index memory used. Inserts take effect immediately; however, rows are not actually deleted until the transaction is committed.
**Transaction parameters.** The next few \[ndbd\] parameters that we discuss are important because they affect the number of parallel transactions and the sizes of transactions that can be handled by the system. \texttt{MaxNoOfConcurrentTransactions} sets the number of parallel transactions possible in a node. \texttt{MaxNoOfConcurrentOperations} sets the number of records that can be in update phase or locked simultaneously.

Both of these parameters (especially \texttt{MaxNoOfConcurrentOperations}) are likely targets for users setting specific values and not using the default value. The default value is set for systems using small transactions, to ensure that these do not use excessive memory.

\texttt{MaxDMLOperationsPerTransaction} sets the maximum number of DML operations that can be performed in a given transaction.

- \texttt{MaxNoOfConcurrentTransactions}

Each cluster data node requires a transaction record for each active transaction in the cluster. The task of coordinating transactions is distributed among all of the data nodes. The total number of transaction records in the cluster is the number of transactions in any given node times the number of nodes in the cluster.

Transaction records are allocated to individual MySQL servers. Each connection to a MySQL server requires at least one transaction record, plus an additional transaction object per table accessed by that connection. This means that a reasonable minimum for the total number of transactions in the cluster can be expressed as

\[
\text{TotalNoOfConcurrentTransactions} = (\text{maximum number of tables accessed in any single transaction} + 1) \times \text{number of SQL nodes}
\]

Suppose that there are 10 SQL nodes using the cluster. A single join involving 10 tables requires 11 transaction records; if there are 10 such joins in a transaction, then \(10 \times 11 = 110\) transaction records are required for this transaction, per MySQL server, or \(110 \times 10 = 1100\) transaction records total. Each data node can be expected to handle \(\text{TotalNoOfConcurrentTransactions} / \text{number of data nodes}\). For an NDB Cluster having 4 data nodes, this would mean setting \texttt{MaxNoOfConcurrentTransactions} on each data node to \(1100 / 4 = 275\). In addition, you should provide for failure recovery by ensuring that a single node group can accommodate all concurrent transactions; in other words, that each data node's \texttt{MaxNoOfConcurrentTransactions} is sufficient to cover a number of transactions equal to \(\text{TotalNoOfConcurrentTransactions} / \text{number of node groups}\). If this cluster has a single node group, then \texttt{MaxNoOfConcurrentTransactions} should be set to 1100 (the same as the total number of concurrent transactions for the entire cluster).

In addition, each transaction involves at least one operation; for this reason, the value set for \texttt{MaxNoOfConcurrentTransactions} should always be no more than the value of \texttt{MaxNoOfConcurrentOperations}.

This parameter must be set to the same value for all cluster data nodes. This is due to the fact that, when a data node fails, the oldest surviving node re-creates the transaction state of all transactions that were ongoing in the failed node.

It is possible to change this value using a rolling restart, but the amount of traffic on the cluster must be such that no more transactions occur than the lower of the old and new levels while this is taking place.

The default value is 4096.

- \texttt{MaxNoOfConcurrentOperations}

It is a good idea to adjust the value of this parameter according to the size and number of transactions. When performing transactions which involve only a few operations and records, the
default value for this parameter is usually sufficient. Performing large transactions involving many records usually requires that you increase its value.

Records are kept for each transaction updating cluster data, both in the transaction coordinator and in the nodes where the actual updates are performed. These records contain state information needed to find UNDO records for rollback, lock queues, and other purposes.

This parameter should be set at a minimum to the number of records to be updated simultaneously in transactions, divided by the number of cluster data nodes. For example, in a cluster which has four data nodes and which is expected to handle one million concurrent updates using transactions, you should set this value to 1000000 / 4 = 250000. To help provide resiliency against failures, it is suggested that you set this parameter to a value that is high enough to permit an individual data node to handle the load for its node group. In other words, you should set the value equal to total number of concurrent operations / number of node groups. (In the case where there is a single node group, this is the same as the total number of concurrent operations for the entire cluster.)

Because each transaction always involves at least one operation, the value of MaxNoOfConcurrentOperations should always be greater than or equal to the value of MaxNoOfConcurrentTransactions.

Read queries which set locks also cause operation records to be created. Some extra space is allocated within individual nodes to accommodate cases where the distribution is not perfect over the nodes.

When queries make use of the unique hash index, there are actually two operation records used per record in the transaction. The first record represents the read in the index table and the second handles the operation on the base table.

The default value is 32768.

This parameter actually handles two values that can be configured separately. The first of these specifies how many operation records are to be placed with the transaction coordinator. The second part specifies how many operation records are to be local to the database.

A very large transaction performed on an eight-node cluster requires as many operation records in the transaction coordinator as there are reads, updates, and deletes involved in the transaction. However, the operation records of the are spread over all eight nodes. Thus, if it is necessary to configure the system for one very large transaction, it is a good idea to configure the two parts separately. MaxNoOfConcurrentOperations is always used to calculate the number of operation records in the transaction coordinator portion of the node.

It is also important to have an idea of the memory requirements for operation records. These consume about 1KB per record.

- MaxNoOfLocalOperations

By default, this parameter is calculated as 1.1 × MaxNoOfConcurrentOperations. This fits systems with many simultaneous transactions, none of them being very large. If there is a need to handle one very large transaction at a time and there are many nodes, it is a good idea to override the default value by explicitly specifying this parameter.

This parameter is deprecated as of NDB 8.0.19, and is subject to removal in a future NDB Cluster release. In addition, this parameter is incompatible with the TransactionMemory parameter; if you try to set values for both parameters in the cluster configuration file (config.ini), the management server refuses to start.
• **MaxDMLOperationsPerTransaction**

This parameter limits the size of a transaction. The transaction is aborted if it requires more than this many DML operations. The minimum number of operations per transaction is 32; however, you can set `MaxDMLOperationsPerTransaction` to 0 to disable any limitation on the number of DML operations per transaction. The maximum (and default) is 4294967295.

The value of this parameter cannot exceed that set for `MaxNoOfConcurrentOperations`.

**Transaction temporary storage.** The next set of `{ndbd}` parameters is used to determine temporary storage when executing a statement that is part of a Cluster transaction. All records are released when the statement is completed and the cluster is waiting for the commit or rollback.

The default values for these parameters are adequate for most situations. However, users with a need to support transactions involving large numbers of rows or operations may need to increase these values to enable better parallelism in the system, whereas users whose applications require relatively small transactions can decrease the values to save memory.

• **MaxNoOfConcurrentIndexOperations**

For queries using a unique hash index, another temporary set of operation records is used during a query's execution phase. This parameter sets the size of that pool of records. Thus, this record is allocated only while executing a part of a query. As soon as this part has been executed, the record is released. The state needed to handle aborts and commits is handled by the normal operation records, where the pool size is set by the parameter `MaxNoOfConcurrentOperations`.

The default value of this parameter is 8192. Only in rare cases of extremely high parallelism using unique hash indexes should it be necessary to increase this value. Using a smaller value is possible and can save memory if the DBA is certain that a high degree of parallelism is not required for the cluster.

This parameter is deprecated as of NDB 8.0.19, and is subject to removal in a future NDB Cluster release. In addition, this parameter is incompatible with the `TransactionMemory` parameter; if you try to set values for both parameters in the cluster configuration file (`config.ini`), the management server refuses to start.

• **MaxNoOfFiredTriggers**

The default value of `MaxNoOfFiredTriggers` is 4000, which is sufficient for most situations. In some cases it can even be decreased if the DBA feels certain the need for parallelism in the cluster is not high.

A record is created when an operation is performed that affects a unique hash index. Inserting or deleting a record in a table with unique hash indexes or updating a column that is part of a unique hash index fires an insert or a delete in the index table. The resulting record is used to represent this index table operation while waiting for the original operation that fired it to complete. This operation is short-lived but can still require a large number of records in its pool for situations with many parallel write operations on a base table containing a set of unique hash indexes.

This parameter is deprecated as of NDB 8.0.19, and is subject to removal in a future NDB Cluster release. In addition, this parameter is incompatible with the `TransactionMemory` parameter; if you try to set values for both parameters in the cluster configuration file (`config.ini`), the management server refuses to start.

• **TransactionBufferMemory**
The memory affected by this parameter is used for tracking operations fired when updating index tables and reading unique indexes. This memory is used to store the key and column information for these operations. It is only very rarely that the value for this parameter needs to be altered from the default.

The default value for **TransactionBufferMemory** is 1MB.

Normal read and write operations use a similar buffer, whose usage is even more short-lived. The compile-time parameter **ZATTRBUF_FILESIZE** (found in `ndb/src/kernel/blocks/Dbtc/Dbtc.hpp`) set to 4000 × 128 bytes (500KB). A similar buffer for key information, **ZDATABUF_FILESIZE** (also in `Dbtc.hpp`) contains 4000 × 16 = 62.5KB of buffer space. **Dbtc** is the module that handles transaction coordination.

**Transaction resource allocation parameters.** The parameters in the following list are used to allocate transaction resources in the transaction coordinator (**DBTC**). Leaving any one of these set to the default (0) dedicates transaction memory for 25% of estimated total data node usage for the corresponding resource. The actual maximum possible values for these parameters are typically limited by the amount of memory available to the data node; setting them has no impact on the total amount of memory allocated to the data node. In addition, you should keep in mind that they control numbers of reserved internal records for the data node independent of any settings for **MaxDMLOperationsPerTransaction**, **MaxNoOfConcurrentIndexOperations**, **MaxNoOfConcurrentOperations**, **MaxNoOfConcurrentScans**, **MaxNoOfConcurrentTransactions**, **MaxNoOfFiredTriggers**, **MaxNoOfLocalScans**, or **TransactionBufferMemory** (see Transaction parameters and Transaction temporary storage).

- **ReservedConcurrentIndexOperations**

  Number of simultaneous index operations having dedicated resources on one data node.

- **ReservedConcurrentOperations**

  Number of simultaneous operations having dedicated resources in transaction coordinators on one data node.

- **ReservedConcurrentScans**

  Number of simultaneous scans having dedicated resources on one data node.

- **ReservedConcurrentTransactions**

  Number of simultaneous transactions having dedicated resources on one data node.

- **ReservedFiredTriggers**

  Number of triggers that have dedicated resources on one ndbd(DB) node.

- **ReservedLocalScans**

  Number of simultaneous fragment scans having dedicated resources on one data node.

- **ReservedTransactionBufferMemory**

  Dynamic buffer space (in bytes) for key and attribute data allocated to each data node.
Defining NDB Cluster Data Nodes

- **TransactionMemory**

  This parameter determines the memory (in bytes) allocated for transactions on each data node. Setting of transaction memory can be handled in any one of the three ways listed here:

  - A number of configuration parameters are incompatible with TransactionMemory. If any of these are set, transaction memory is calculated as it was previous to NDB 8.0.19. You should be aware that it is not possible to set any of these parameters concurrently with TransactionMemory; if you attempt to do so, the management server is unable to start (see Parameters incompatible with TransactionMemory).

  - If TransactionMemory is set, this value is used for determining transaction memory.

  - If neither any incompatible parameters are set nor TransactionMemory is set, transaction memory is set by NDB to 10% of the value of the DataMemory configuration parameter.

**Parameters incompatible with TransactionMemory.** The following parameters cannot be used concurrently with TransactionMemory and are deprecated as of NDB 8.0.19:

- MaxNoOfConcurrentIndexOperations
- MaxNoOfFiredTriggers
- MaxNoOfLocalOperations
- MaxNoOfLocalScans

Explicitly setting any of the parameters just listed when TransactionMemory has also been set in the cluster configuration file (config.ini) keeps the management node from starting.

For more information regarding resource allocation in NDB Cluster data nodes, see Section 5.3.13, “Data Node Memory Management”.

**Scans and buffering.** There are additional [ndbd] parameters in the Dblqh module (in ndb/src/kernel/blocks/Dblqh/Dblqh.hpp) that affect reads and updates. These include ZATTRINBUF_FILESIZE, set by default to 10000 × 128 bytes (1250KB) and ZDATABUF_FILE_SIZE, set by default to 10000*16 bytes (roughly 156KB) of buffer space. To date, there have been neither any reports from users nor any results from our own extensive tests suggesting that either of these compile-time limits should be increased.

- BatchSizePerLocalScan

  This parameter is used to calculate the number of lock records used to handle concurrent scan operations.

  **BatchSizePerLocalScan** has a strong connection to the **BatchSize** defined in the SQL nodes.

  Deprecated as of NDB 8.0.19.

- LongMessageBuffer

  This is an internal buffer used for passing messages within individual nodes and between nodes. The default is 64MB.

  This parameter seldom needs to be changed from the default.
Defining NDB Cluster Data Nodes

Maximum scan batch size used for building foreign keys. Increasing the value set for this parameter may speed up building of foreign key builds at the expense of greater impact to ongoing traffic.

- **MaxNoOfConcurrentScans**

  This parameter is used to control the number of parallel scans that can be performed in the cluster. Each transaction coordinator can handle the number of parallel scans defined for this parameter. Each scan query is performed by scanning all partitions in parallel. Each partition scan uses a scan record in the node where the partition is located, the number of records being the value of this parameter times the number of nodes. The cluster should be able to sustain MaxNoOfConcurrentScans scans concurrently from all nodes in the cluster.

  Scans are actually performed in two cases. The first of these cases occurs when no hash or ordered indexes exists to handle the query, in which case the query is executed by performing a full table scan. The second case is encountered when there is no hash index to support the query but there is an ordered index. Using the ordered index means executing a parallel range scan. The order is kept on the local partitions only, so it is necessary to perform the index scan on all partitions.

  The default value of MaxNoOfConcurrentScans is 256. The maximum value is 500.

- **MaxNoOfLocalScans**

  Specifies the number of local scan records if many scans are not fully parallelized. When the number of local scan records is not provided, it is calculated as shown here:

  \[
  4 \times \text{MaxNoOfConcurrentScans} \times \# \text{data nodes} + 2
  \]

  This parameter is deprecated as of NDB 8.0.19, and is subject to removal in a future NDB Cluster release. In addition, this parameter is incompatible with the TransactionMemory parameter; if you try to set values for both parameters in the cluster configuration file (config.ini), the management server refuses to start.

- **MaxParallelCopyInstances**

  This parameter sets the parallelization used in the copy phase of a node restart or system restart, when a node that is currently just starting is synchronised with a node that already has current data by copying over any changed records from the node that is up to date. Because full parallelism in such cases can lead to overload situations, MaxParallelCopyInstances provides a means to decrease it. This parameter's default value 0. This value means that the effective parallelism is equal to the number of LDM instances in the node just starting as well as the node updating it.

- **MaxParallelScansPerFragment**

  It is possible to configure the maximum number of parallel scans (TUP scans and TUX scans) allowed before they begin queuing for serial handling. You can increase this to take advantage of any unused CPU when performing large number of scans in parallel and improve their performance.

- **MaxReorgBuildBatchSize**

  Maximum scan batch size used for reorganization of table partitions. Increasing the value set for this parameter may speed up reorganization at the expense of greater impact to ongoing traffic.

- **MaxUIBuildBatchSize**

  Maximum scan batch size used for building unique keys. Increasing the value set for this parameter may speed up such builds at the expense of greater impact to ongoing traffic.
Memory Allocation

MaxAllocate

This is the maximum size of the memory unit to use when allocating memory for tables. In cases where NDB gives Out of memory errors, but it is evident by examining the cluster logs or the output of DUMP 1000 that all available memory has not yet been used, you can increase the value of this parameter (or MaxNoOfTables, or both) to cause NDB to make sufficient memory available.

Multiple Transporters

Beginning with version 8.0.20, NDB allocates multiple transporters for communication between pairs of data nodes. The number of transporters so allocated can be influenced by setting an appropriate value for the NodeGroupTransporters parameter introduced in that release.

NodeGroupTransporters

This parameter determines the number of transporters used between nodes in the same node group. The default value (0) means that the number of transporters used is the same as the number of LDMs in the node. This should be sufficient for most use cases; thus it should seldom be necessary to change this value from its default.

Setting NodeGroupTransporters to a number greater than the number of LDM threads or the number of TC threads, whichever is higher, causes NDB to use the maximum of these two numbers of threads. This means that a value greater than this is effectively ignored.

Hash Map Size

DefaultHashMapSize

The original intended use for this parameter was to facilitate upgrades and especially downgrades to and from very old releases with differing default hash map sizes. This is not an issue when upgrading from NDB Cluster 7.3 (or later) to later versions.

Decreasing this parameter online after any tables have been created or modified with DefaultHashMapSize equal to 3840 is not currently supported.

Logging and checkpointing.

The following [ndbd] parameters control log and checkpoint behavior.

- FragmentLogFileSize

Setting this parameter enables you to control directly the size of redo log files. This can be useful in situations when NDB Cluster is operating under a high load and it is unable to close fragment log files quickly enough before attempting to open new ones (only 2 fragment log files can be open at one time); increasing the size of the fragment log files gives the cluster more time before having to open each new fragment log file. The default value for this parameter is 16M.

For more information about fragment log files, see the description for NoOfFragmentLogFiles.

- InitialNoOfOpenFiles

This parameter sets the initial number of internal threads to allocate for open files.

The default value is 27.

- InitFragmentLogFiles
By default, fragment log files are created sparsely when performing an initial start of a data
node—that is, depending on the operating system and file system in use, not all bytes are
necessarily written to disk. However, it is possible to override this behavior and force all bytes to
be written, regardless of the platform and file system type being used, by means of this parameter.
InitFragmentLogFiles takes either of two values:

- **SPARSE**. Fragment log files are created sparsely. This is the default value.
- **FULL**. Force all bytes of the fragment log file to be written to disk.

Depending on your operating system and file system, setting InitFragmentLogFiles=FULL may
help eliminate I/O errors on writes to the REDO log.

- **EnablePartialLcp**

When `true`, enable partial local checkpoints: This means that each LCP records only part of the full
database, plus any records containing rows changed since the last LCP; if no rows have changed,
the LCP updates only the LCP control file and does not update any data files.

If EnablePartialLcp is disabled (`false`), each LCP uses only a single file and writes a full
checkpoint; this requires the least amount of disk space for LCPs, but increases the write load for
each LCP. The default value is enabled (`true`). The proportion of space used by partial LCPS can
be modified by the setting for the `RecoveryWork` configuration parameter.

For more information about files and directories used for full and partial LCPS, see NDB Cluster Data

Setting this parameter to `false` also disables the calculation of disk write speed used by the
adaptive LCP control mechanism.

- **LcpScanProgressTimeout**

A local checkpoint fragment scan watchdog checks periodically for no progress in each fragment
scan performed as part of a local checkpoint, and shuts down the node if there is no progress after a
given amount of time has elapsed. This interval can be set using the LcpScanProgressTimeout
data node configuration parameter, which sets the maximum time for which the local checkpoint can
be stalled before the LCP fragment scan watchdog shuts down the node.

The default value is 60 seconds (providing compatibility with previous releases). Setting this
parameter to 0 disables the LCP fragment scan watchdog altogether.

- **MaxNoOfOpenFiles**

This parameter sets a ceiling on how many internal threads to allocate for open files. *Any situation
requiring a change in this parameter should be reported as a bug.*

The default value is 0. However, the minimum value to which this parameter can be set is 20.

- **MaxNoOfSavedMessages**

This parameter sets the maximum number of errors written in the error log as well as the maximum
number of trace files that are kept before overwriting the existing ones. Trace files are generated
when, for whatever reason, the node crashes.

The default is 25, which sets these maximums to 25 error messages and 25 trace files.

- **MaxLCPStartDelay**
In parallel data node recovery, only table data is actually copied and synchronized in parallel; synchronization of metadata such as dictionary and checkpoint information is done in a serial fashion. In addition, recovery of dictionary and checkpoint information cannot be executed in parallel with performing of local checkpoints. This means that, when starting or restarting many data nodes concurrently, data nodes may be forced to wait while a local checkpoint is performed, which can result in longer node recovery times.

It is possible to force a delay in the local checkpoint to permit more (and possibly all) data nodes to complete metadata synchronization; once each data node’s metadata synchronization is complete, all of the data nodes can recover table data in parallel, even while the local checkpoint is being executed. To force such a delay, set \texttt{MaxLCPStartDelay}, which determines the number of seconds the cluster can wait to begin a local checkpoint while data nodes continue to synchronize metadata. This parameter should be set in the \texttt{[ndbd default]} section of the \texttt{config.ini} file, so that it is the same for all data nodes. The maximum value is 600; the default is 0.

- \textbf{NoOfFragmentLogFiles}

This parameter sets the number of REDO log files for the node, and thus the amount of space allocated to REDO logging. Because the REDO log files are organized in a ring, it is extremely important that the first and last log files in the set (sometimes referred to as the “head” and “tail” log files, respectively) do not meet. When these approach one another too closely, the node begins aborting all transactions encompassing updates due to a lack of room for new log records.

A \texttt{REDO} log record is not removed until both required local checkpoints have been completed since that log record was inserted. Checkpointing frequency is determined by its own set of configuration parameters discussed elsewhere in this chapter.

The default parameter value is 16, which by default means 16 sets of 4 16MB files for a total of 1024MB. The size of the individual log files is configurable using the \texttt{FragmentLogFileSize} parameter. In scenarios requiring a great many updates, the value for \texttt{NoOfFragmentLogFiles} may need to be set as high as 300 or even higher to provide sufficient space for REDO logs.

If the checkpointing is slow and there are so many writes to the database that the log files are full and the log tail cannot be cut without jeopardizing recovery, all updating transactions are aborted with internal error code 410 (\texttt{Out of log file space temporarily}). This condition prevails until a checkpoint has completed and the log tail can be moved forward.

\textbf{Important}

This parameter cannot be changed “on the fly”; you must restart the node using \texttt{--initial}. If you wish to change this value for all data nodes in a running cluster, you can do so using a rolling node restart (using \texttt{--initial} when starting each data node).
Defining NDB Cluster Data Nodes

- **RecoveryWork**

  Percentage of storage overhead for LCP files. This parameter has an effect only when `EnablePartialLcp` is true, that is, only when partial local checkpoints are enabled. A higher value means:

  - Fewer records are written for each LCP, LCPs use more space
  - More work is needed during restarts

  A lower value for `RecoveryWork` means:

  - More records are written during each LCP, but LCPs require less space on disk.
  - Less work during restart and thus faster restarts, at the expense of more work during normal operations

  For example, setting `RecoveryWork` to 60 means that the total size of an LCP is roughly $1 + 0.6 = 1.6$ times the size of the data to be checkpointed. This means that 60% more work is required during the restore phase of a restart compared to the work done during a restart that uses full checkpoints. (This is more than compensated for during other phases of the restart such that the restart as a whole is still faster when using partial LCPs than when using full LCPs.) In order not to fill up the redo log, it is necessary to write at $1 + \left( \frac{1}{\text{RecoveryWork}} \right)$ times the rate of data changes during checkpoints—thus, when `RecoveryWork` = 60, it is necessary to write at approximately $1 + \left( \frac{1}{0.6} \right) = 2.67$ times the change rate. In other words, if changes are being written at 10 MB/second, the checkpoint needs to be written at roughly 26.7 MB/second.

  Setting `RecoveryWork` = 40 means that only 1.4 times the total LCP size is needed (and thus the restore phase takes 10 to 15 percent less time. In this case, the checkpoint write rate is 3.5 times the rate of change.

  The NDB source distribution includes a test program for simulating LCPs. `lcp_simulator.cc` can be found in `storage/ndb/src/kernel/blocks/backup/`. To compile and run it on Unix platforms, execute the commands shown here:

  ```
  shell> gcc lcp_simulator.cc
  shell> ./a.out
  ```

  This program has no dependencies other than `stdio.h`, and does not require a connection to an NDB cluster or a MySQL server. By default, it simulates 300 LCPs (three sets of 100 LCPs, each consisting of inserts, updates, and deletes, in turn), reporting the size of the LCP after each one. You can alter the simulation by changing the values of `recovery_work`, `insert_work`, and `delete_work` in the source and recompiling. For more information, see the source of the program.

- **InsertRecoveryWork**

  Percentage of `RecoveryWork` used for inserted rows. A higher value increases the number of writes during a local checkpoint, and decreases the total size of the LCP. A lower value decreases the number of writes during an LCP, but results in more space being used for the LCP, which means that recovery takes longer. This parameter has an effect only when `EnablePartialLcp` is true, that is, only when partial local checkpoints are enabled.

- **EnableRedoControl**

  Enable adaptive checkpointing speed for controlling redo log usage. Set to `false` to disable (the default). Setting `EnablePartialLcp` to `false` also disables the adaptive calculation.

  When enabled, `EnableRedoControl` allows the data nodes greater flexibility with regard to the rate at which they write LCPs to disk. More specifically, enabling this parameter means that higher
write rates can be employed, so that LCPs can complete and Redo logs be trimmed more quickly, thereby reducing recovery time and disk space requirements. This functionality allows data nodes to make better use of the higher rate of I/O and greater bandwidth available from modern solid-state storage devices and protocols, such as solid-state drives (SSDs) using Non-Volatile Memory Express (NVMe).

The parameter currently defaults to `false` (disabled) due to the fact that NDB is still deployed widely on systems whose I/O or bandwidth is constrained relative to those employing solid-state technology, such as those using conventional hard disks (HDDs). In settings such as these, the `EnableRedoControl` mechanism can easily cause the I/O subsystem to become saturated, increasing wait times for data node input and output. In particular, this can cause issues with NDB Disk Data tables which have tablespaces or log file groups sharing a constrained IO subsystem with data node LCP and redo log files; such problems potentially include node or cluster failure due to GCP stop errors.

**Metadata objects.** The next set of `[ndbd]` parameters defines pool sizes for metadata objects, used to define the maximum number of attributes, tables, indexes, and trigger objects used by indexes, events, and replication between clusters.

**Note**

These act merely as “suggestions” to the cluster, and any that are not specified revert to the default values shown.

- `MaxNoOfAttributes`

This parameter sets a suggested maximum number of attributes that can be defined in the cluster; like `MaxNoOfTables`, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than `MaxNoOfAttributes` attributes.)

The default value is 1000, with the minimum possible value being 32. The maximum is 4294967039. Each attribute consumes around 200 bytes of storage per node due to the fact that all metadata is fully replicated on the servers.

When setting `MaxNoOfAttributes`, it is important to prepare in advance for any `ALTER TABLE` statements that you might want to perform in the future. This is due to the fact, during the execution of `ALTER TABLE` on a Cluster table, 3 times the number of attributes as in the original table are used, and a good practice is to permit double this amount. For example, if the NDB Cluster table having the greatest number of attributes (`greatest_number_of_attributes`) has 100 attributes, a good starting point for the value of `MaxNoOfAttributes` would be `6 * greatest_number_of_attributes = 600`.

You should also estimate the average number of attributes per table and multiply this by `MaxNoOfTables`. If this value is larger than the value obtained in the previous paragraph, you should use the larger value instead.

Assuming that you can create all desired tables without any problems, you should also verify that this number is sufficient by trying an actual `ALTER TABLE` after configuring the parameter. If this is not successful, increase `MaxNoOfAttributes` by another multiple of `MaxNoOfTables` and test it again.

- `MaxNoOfTables`
A table object is allocated for each table and for each unique hash index in the cluster. This parameter sets a suggested maximum number of table objects for the cluster as a whole; like MaxNoOfAttributes, it is not intended to function as a hard upper limit.

(In older NDB Cluster releases, this parameter was sometimes treated as a hard limit for certain operations. This caused problems with NDB Cluster Replication, when it was possible to create more tables than could be replicated, and sometimes led to confusion when it was possible [or not possible, depending on the circumstances] to create more than MaxNoOfTables tables.)

For each attribute that has a BLOB data type an extra table is used to store most of the BLOB data. These tables also must be taken into account when defining the total number of tables.

The default value of this parameter is 128. The minimum is 8 and the maximum is 20320. Each table object consumes approximately 20KB per node.

*MaxNoOfOrderedIndexes*

For each ordered index in the cluster, an object is allocated describing what is being indexed and its storage segments. By default, each index so defined also defines an ordered index. Each unique index and primary key has both an ordered index and a hash index. MaxNoOfOrderedIndexes sets the total number of ordered indexes that can be in use in the system at any one time.

The default value of this parameter is 128. Each index object consumes approximately 10KB of data per node.

*MaxNoOfUniqueHashIndexes*

For each unique index that is not a primary key, a special table is allocated that maps the unique key to the primary key of the indexed table. By default, an ordered index is also defined for each unique index. To prevent this, you must specify the USING HASH option when defining the unique index.

The default value is 64. Each index consumes approximately 15KB per node.
Defining NDB Cluster Data Nodes

- **MaxNoOfTriggers**

  Internal update, insert, and delete triggers are allocated for each unique hash index. (This means that three triggers are created for each unique hash index.) However, an *ordered* index requires only a single trigger object. Backups also use three trigger objects for each normal table in the cluster.

  Replication between clusters also makes use of internal triggers.

  This parameter sets the maximum number of trigger objects in the cluster.

  The default value is 768.

- **MaxNoOfSubscriptions**

  Each NDB table in an NDB Cluster requires a subscription in the NDB kernel. For some NDB API applications, it may be necessary or desirable to change this parameter. However, for normal usage with MySQL servers acting as SQL nodes, there is not any need to do so.

  The default value for MaxNoOfSubscriptions is 0, which is treated as equal to MaxNoOfTables. Each subscription consumes 108 bytes.

- **MaxNoOfSubscribers**

  This parameter is of interest only when using NDB Cluster Replication. The default value is 0. Prior to NDB 8.0.25, this was treated as 2 * MaxNoOfTables; beginning with NDB 8.0.25, it is treated as 2 * MaxNoOfTables + 2 * [number of API nodes]. There is one subscription per NDB table for each of two MySQL servers (one acting as the replication source and the other as the replica). Each subscriber uses 16 bytes of memory.

  When using circular replication, multi-source replication, and other replication setups involving more than 2 MySQL servers, you should increase this parameter to the number of mysqld processes included in replication (this is often, but not always, the same as the number of clusters). For example, if you have a circular replication setup using three NDB Clusters, with one mysqld attached to each cluster, and each of these mysqld processes acts as a source and as a replica, you should set MaxNoOfSubscribers equal to 3 * MaxNoOfTables.

  For more information, see Chapter 8, NDB Cluster Replication.

- **MaxNoOfConcurrentSubOperations**

  This parameter sets a ceiling on the number of operations that can be performed by all API nodes in the cluster at one time. The default value (256) is sufficient for normal operations, and might need to be adjusted only in scenarios where there are a great many API nodes each performing a high volume of operations concurrently.

**Boolean parameters.** The behavior of data nodes is also affected by a set of [ndbd] parameters taking on boolean values. These parameters can each be specified as **TRUE** by setting them equal to 1 or Y, and as **FALSE** by setting them equal to 0 or N.

- **CompressedLCP**

  Setting this parameter to 1 causes local checkpoint files to be compressed. The compression used is equivalent to gzip --fast, and can save 50% or more of the space required on the data node to store uncompressed checkpoint files. Compressed LCPs can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the [ndbd default] section of the config.ini file).
Defining NDB Cluster Data Nodes

**Important**

You cannot restore a compressed local checkpoint to a cluster running a MySQL version that does not support this feature.

The default value is 0 (disabled).

- **CrashOnCorruptedTuple**

When this parameter is enabled (the default), it forces a data node to shut down whenever it encounters a corrupted tuple.

- **Diskless**

It is possible to specify NDB Cluster tables as diskless, meaning that tables are not checkpointed to disk and that no logging occurs. Such tables exist only in main memory. A consequence of using diskless tables is that neither the tables nor the records in those tables survive a crash. However, when operating in diskless mode, it is possible to run ndbd on a diskless computer.

**Important**

This feature causes the entire cluster to operate in diskless mode.

When this feature is enabled, Cluster online backup is disabled. In addition, a partial start of the cluster is not possible.

*Diskless* is disabled by default.

- **LateAlloc**

Allocate memory for this data node after a connection to the management server has been established. Enabled by default.

- **LockPagesInMainMemory**

For a number of operating systems, including Solaris and Linux, it is possible to lock a process into memory and so avoid any swapping to disk. This can be used to help guarantee the cluster’s real-time characteristics.

This parameter takes one of the integer values 0, 1, or 2, which act as shown in the following list:

- **0**: Disables locking. This is the default value.
- **1**: Performs the lock after allocating memory for the process.
- **2**: Performs the lock before memory for the process is allocated.

If the operating system is not configured to permit unprivileged users to lock pages, then the data node process making use of this parameter may have to be run as system root. (*LockPagesInMainMemory* uses the *mlockall* function. From Linux kernel 2.6.9, unprivileged users can lock memory as limited by *max locked memory*. For more information, see `ulimit -l` and [http://linux.die.net/man/2/mlock](http://linux.die.net/man/2/mlock).

**Note**

In older NDB Cluster releases, this parameter was a Boolean. 0 or `false` was the default setting, and disabled locking. 1 or `true` enabled locking of
Defining NDB Cluster Data Nodes

the process after its memory was allocated. NDB Cluster 8.0 treats `true` or `false` for the value of this parameter as an error.

**Important**

Beginning with glibc 2.10, glibc uses per-thread arenas to reduce lock contention on a shared pool, which consumes real memory. In general, a data node process does not need per-thread arenas, since it does not perform any memory allocation after startup. (This difference in allocators does not appear to affect performance significantly.)

The glibc behavior is intended to be configurable via the `MALLOC_ARENA_MAX` environment variable, but a bug in this mechanism prior to glibc 2.16 meant that this variable could not be set to less than 8, so that the wasted memory could not be reclaimed. (Bug #15907219; see also [http://sourceware.org/bugzilla/show_bug.cgi?id=13137](http://sourceware.org/bugzilla/show_bug.cgi?id=13137) for more information concerning this issue.)

One possible workaround for this problem is to use the `LD_PRELOAD` environment variable to preload a jemalloc memory allocation library to take the place of that supplied with glibc.

- **ODirect**

  Enabling this parameter causes NDB to attempt using `O_DIRECT` writes for LCP, backups, and redo logs, often lowering `kswapd` and CPU usage. When using NDB Cluster on Linux, enable `ODirect` if you are using a 2.6 or later kernel.

  `ODirect` is disabled by default.

- **ODirectSyncFlag**

  When this parameter is enabled, redo log writes are performed such that each completed file system write is handled as a call to `fsync`. The setting for this parameter is ignored if at least one of the following conditions is true:

  - `ODirect` is not enabled.
  - `InitFragmentLogFiles` is set to `SPARSE`.

  Disabled by default.

- **RestartOnErrorInsert**

  This feature is accessible only when building the debug version where it is possible to insert errors in the execution of individual blocks of code as part of testing.

  This feature is disabled by default.

- **StopOnError**

  This parameter specifies whether a data node process should exit or perform an automatic restart when an error condition is encountered.

  This parameter's default value is 1; this means that, by default, an error causes the data node process to halt.

  When an error is encountered and `StopOnError` is 0, the data node process is restarted.
Users of MySQL Cluster Manager should note that, when `StopOnError` equals 1, this prevents the MySQL Cluster Manager agent from restarting any data nodes after it has performed its own restart and recovery. See Starting and Stopping the Agent on Linux, for more information.

- **UseShm**

Enable a shared memory connection between this data node and the API node also running on this host. Set to 1 to enable.

**Controlling Timeouts, Intervals, and Disk Paging**

There are a number of `ndbd` parameters specifying timeouts and intervals between various actions in Cluster data nodes. Most of the timeout values are specified in milliseconds. Any exceptions to this are mentioned where applicable.

- **TimeBetweenWatchDogCheck**

  To prevent the main thread from getting stuck in an endless loop at some point, a “watchdog” thread checks the main thread. This parameter specifies the number of milliseconds between checks. If the process remains in the same state after three checks, the watchdog thread terminates it.

  This parameter can easily be changed for purposes of experimentation or to adapt to local conditions. It can be specified on a per-node basis although there seems to be little reason for doing so.

  The default timeout is 6000 milliseconds (6 seconds).

- **TimeBetweenWatchDogCheckInitial**

  This is similar to the `TimeBetweenWatchDogCheck` parameter, except that `TimeBetweenWatchDogCheckInitial` controls the amount of time that passes between execution checks inside a storage node in the early start phases during which memory is allocated.

  The default timeout is 6000 milliseconds (6 seconds).

- **StartPartialTimeout**

  This parameter specifies how long the Cluster waits for all data nodes to come up before the cluster initialization routine is invoked. This timeout is used to avoid a partial Cluster startup whenever possible.

  This parameter is overridden when performing an initial start or initial restart of the cluster.

  The default value is 30000 milliseconds (30 seconds). 0 disables the timeout, in which case the cluster may start only if all nodes are available.

- **StartPartitionedTimeout**

  If the cluster is ready to start after waiting for `StartPartialTimeout` milliseconds but is still possibly in a partitioned state, the cluster waits until this timeout has also passed. If `StartPartitionedTimeout` is set to 0, the cluster waits indefinitely (2^{32}−1 ms, or approximately 49.71 days).

  This parameter is overridden when performing an initial start or initial restart of the cluster.

- **StartFailureTimeout**
If a data node has not completed its startup sequence within the time specified by this parameter, the node startup fails. Setting this parameter to 0 (the default value) means that no data node timeout is applied.

For nonzero values, this parameter is measured in milliseconds. For data nodes containing extremely large amounts of data, this parameter should be increased. For example, in the case of a data node containing several gigabytes of data, a period as long as $10^{-15}$ minutes (that is, 600000 to 1000000 milliseconds) might be required to perform a node restart.

- **StartNoNodeGroupTimeout**

When a data node is configured with `Nodegroup = 65536`, is regarded as not being assigned to any node group. When that is done, the cluster waits `StartNoNodegroupTimeout` milliseconds, then treats such nodes as though they had been added to the list passed to the `--nowait-nodes` option, and starts. The default value is `15000` (that is, the management server waits 15 seconds). Setting this parameter equal to 0 means that the cluster waits indefinitely.

`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.

See Section 7.7, “Adding NDB Cluster Data Nodes Online”, for more information.

- **HeartbeatIntervalDbDb**

One of the primary methods of discovering failed nodes is by the use of heartbeats. This parameter states how often heartbeat signals are sent and how often to expect to receive them. Heartbeats cannot be disabled.

After missing four heartbeat intervals in a row, the node is declared dead. Thus, the maximum time for discovering a failure through the heartbeat mechanism is five times the heartbeat interval.

The default heartbeat interval is 5000 milliseconds (5 seconds). This parameter must not be changed drastically and should not vary widely between nodes. If one node uses 5000 milliseconds and the node watching it uses 1000 milliseconds, obviously the node is declared dead very quickly. This parameter can be changed during an online software upgrade, but only in small increments.

See also Network communication and latency, as well as the description of the `ConnectCheckIntervalDelay` configuration parameter.

- **HeartbeatIntervalDbApi**

Each data node sends heartbeat signals to each MySQL server (SQL node) to ensure that it remains in contact. If a MySQL server fails to send a heartbeat in time it is declared “dead,” in which case all ongoing transactions are completed and all resources released. The SQL node cannot reconnect until all activities initiated by the previous MySQL instance have been completed. The three-heartbeat criteria for this determination are the same as described for `HeartbeatIntervalDbDb`.

The default interval is 1500 milliseconds (1.5 seconds). This interval can vary between individual data nodes because each data node watches the MySQL servers connected to it, independently of all other data nodes.

For more information, see Network communication and latency.
Data nodes send heartbeats to one another in a circular fashion whereby each data node monitors the previous one. If a heartbeat is not detected by a given data node, this node declares the previous data node in the circle “dead” (that is, no longer accessible by the cluster). The determination that a data node is dead is done globally; in other words; once a data node is declared dead, it is regarded as such by all nodes in the cluster.

It is possible for heartbeats between data nodes residing on different hosts to be too slow compared to heartbeats between other pairs of nodes (for example, due to a very low heartbeat interval or temporary connection problem), such that a data node is declared dead, even though the node can still function as part of the cluster. In this type of situation, it may be that the order in which heartbeats are transmitted between data nodes makes a difference as to whether or not a particular data node is declared dead. If this declaration occurs unnecessarily, this can in turn lead to the unnecessary loss of a node group and as thus to a failure of the cluster.

Consider a setup where there are 4 data nodes A, B, C, and D running on 2 host computers host1 and host2, and that these data nodes make up 2 node groups, as shown in the following table:

<table>
<thead>
<tr>
<th>Node Group</th>
<th>Nodes Running on host1</th>
<th>Nodes Running on host2</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node Group 0:</td>
<td>Node A</td>
<td>Node B</td>
</tr>
<tr>
<td>Node Group 1:</td>
<td>Node C</td>
<td>Node D</td>
</tr>
</tbody>
</table>

Suppose the heartbeats are transmitted in the order A->B->C->D->A. In this case, the loss of the heartbeat between the hosts causes node B to declare node A dead and node C to declare node B dead. This results in loss of Node Group 0, and so the cluster fails. On the other hand, if the order of transmission is A->B->D->C->A (and all other conditions remain as previously stated), the loss of the heartbeat causes nodes A and D to be declared dead; in this case, each node group has one surviving node, and the cluster survives.

The `HeartbeatOrder` configuration parameter makes the order of heartbeat transmission user-configurable. The default value for `HeartbeatOrder` is zero; allowing the default value to be used on all data nodes causes the order of heartbeat transmission to be determined by NDB. If this parameter is used, it must be set to a nonzero value (maximum 65535) for every data node in the cluster, and this value must be unique for each data node; this causes the heartbeat transmission to proceed from data node to data node in the order of their `HeartbeatOrder` values from lowest to highest (and then directly from the data node having the highest `HeartbeatOrder` to the data node having the lowest value, to complete the circle). The values need not be consecutive. For example, to force the heartbeat transmission order A->B->D->C->A in the scenario outlined previously, you could set the `HeartbeatOrder` values as shown here:

<table>
<thead>
<tr>
<th>Node</th>
<th><code>HeartbeatOrder</code> Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>A</td>
<td>10</td>
</tr>
<tr>
<td>B</td>
<td>20</td>
</tr>
<tr>
<td>C</td>
<td>30</td>
</tr>
<tr>
<td>D</td>
<td>25</td>
</tr>
</tbody>
</table>

To use this parameter to change the heartbeat transmission order in a running NDB Cluster, you must first set `HeartbeatOrder` for each data node in the cluster in the global configuration.
Defining NDB Cluster Data Nodes

(config.ini) file (or files). To cause the change to take effect, you must perform either of the following:

- A complete shutdown and restart of the entire cluster.
- 2 rolling restarts of the cluster in succession. *All nodes must be restarted in the same order in both rolling restarts.*

You can use DUMP 908 to observe the effect of this parameter in the data node logs.

- **ConnectCheckIntervalDelay**

This parameter enables connection checking between data nodes after one of them has failed heartbeat checks for 5 intervals of up to HeartbeatIntervalDbDb milliseconds.

Such a data node that further fails to respond within an interval of ConnectCheckIntervalDelay milliseconds is considered suspect, and is considered dead after two such intervals. This can be useful in setups with known latency issues.

The default value for this parameter is 0 (disabled).

- **TimeBetweenLocalCheckpoints**

This parameter is an exception in that it does not specify a time to wait before starting a new local checkpoint; rather, it is used to ensure that local checkpoints are not performed in a cluster where relatively few updates are taking place. In most clusters with high update rates, it is likely that a new local checkpoint is started immediately after the previous one has been completed.

The size of all write operations executed since the start of the previous local checkpoints is added. This parameter is also exceptional in that it is specified as the base-2 logarithm of the number of 4-byte words, so that the default value 20 means 4MB (4 × 2^20) of write operations, 21 would mean 8MB, and so on up to a maximum value of 31, which equates to 8GB of write operations.

All the write operations in the cluster are added together. Setting TimeBetweenLocalCheckpoints to 6 or less means that local checkpoints are executed continuously without pause, independent of the cluster’s workload.

- **TimeBetweenGlobalCheckpoints**

When a transaction is committed, it is committed in main memory in all nodes on which the data is mirrored. However, transaction log records are not flushed to disk as part of the commit. The reasoning behind this behavior is that having the transaction safely committed on at least two autonomous host machines should meet reasonable standards for durability.

It is also important to ensure that even the worst of cases—a complete crash of the cluster—is handled properly. To guarantee that this happens, all transactions taking place within a given interval are put into a global checkpoint, which can be thought of as a set of committed transactions that has been flushed to disk. In other words, as part of the commit process, a transaction is placed in a global checkpoint group. Later, this group's log records are flushed to disk, and then the entire group of transactions is safely committed to disk on all computers in the cluster.

In NDB 8.0.19 and later, it recommended when using solid-state disks (especially those employing NVMe) with Disk Data tables that you reduce this value. In such cases, you should also ensure that MaxDiskDataLatency is set to a proper level.

This parameter defines the interval between global checkpoints. The default is 2000 milliseconds.

- **TimeBetweenGlobalCheckpointsTimeout**
This parameter defines the minimum timeout between global checkpoints. The default is 120000 milliseconds.

- **TimeBetweenEpochs**

This parameter defines the interval between synchronization epochs for NDB Cluster Replication. The default value is 100 milliseconds.

`TimeBetweenEpochs` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

- **TimeBetweenEpochsTimeout**

This parameter defines a timeout for synchronization epochs for NDB Cluster Replication. If a node fails to participate in a global checkpoint within the time determined by this parameter, the node is shut down. The default value is 0; in other words, the timeout is disabled.

`TimeBetweenEpochsTimeout` is part of the implementation of “micro-GCPs”, which can be used to improve the performance of NDB Cluster Replication.

The current value of this parameter and a warning are written to the cluster log whenever a GCP save takes longer than 1 minute or a GCP commit takes longer than 10 seconds.

Setting this parameter to zero has the effect of disabling GCP stops caused by save timeouts, commit timeouts, or both. The maximum possible value for this parameter is 256000 milliseconds.

- **MaxBufferedEpochs**

The number of unprocessed epochs by which a subscribing node can lag behind. Exceeding this number causes a lagging subscriber to be disconnected.

The default value of 100 is sufficient for most normal operations. If a subscribing node does lag enough to cause disconnections, it is usually due to network or scheduling issues with regard to processes or threads. (In rare circumstances, the problem may be due to a bug in the NDB client.) It may be desirable to set the value lower than the default when epochs are longer.

Disconnection prevents client issues from affecting the data node service, running out of memory to buffer data, and eventually shutting down. Instead, only the client is affected as a result of the disconnect (by, for example gap events in the binary log), forcing the client to reconnect or restart the process.

- **MaxBufferedEpochBytes**

The total number of bytes allocated for buffering epochs by this node.

- **TimeBetweenInactiveTransactionAbortCheck**

Timeout handling is performed by checking a timer on each transaction once for every interval specified by this parameter. Thus, if this parameter is set to 1000 milliseconds, every transaction is checked for timing out once per second.

The default value is 1000 milliseconds (1 second).
• **TransactionInactiveTimeout**

This parameter states the maximum time that is permitted to lapse between operations in the same transaction before the transaction is aborted.

The default for this parameter is \(4G\) (also the maximum). For a real-time database that needs to ensure that no transaction keeps locks for too long, this parameter should be set to a relatively small value. Setting it to 0 means that the application never times out. The unit is milliseconds.

• **TransactionDeadlockDetectionTimeout**

When a node executes a query involving a transaction, the node waits for the other nodes in the cluster to respond before continuing. This parameter sets the amount of time that the transaction can spend executing within a data node, that is, the time that the transaction coordinator waits for each data node participating in the transaction to execute a request.

A failure to respond can occur for any of the following reasons:

• The node is “dead”
• The operation has entered a lock queue
• The node requested to perform the action could be heavily overloaded.

This timeout parameter states how long the transaction coordinator waits for query execution by another node before aborting the transaction, and is important for both node failure handling and deadlock detection.

The default timeout value is 1200 milliseconds (1.2 seconds).

The minimum for this parameter is 50 milliseconds.

• **DiskSyncSize**

This is the maximum number of bytes to store before flushing data to a local checkpoint file. This is done to prevent write buffering, which can impede performance significantly. This parameter is not intended to take the place of `TimeBetweenLocalCheckpoints`.

```
Note
```

When `ODirect` is enabled, it is not necessary to set `DiskSyncSize`; in fact, in such cases its value is simply ignored.

The default value is 4M (4 megabytes).

• **MaxDiskWriteSpeed**

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when no restarts (by this data node or any other data node) are taking place in this NDB Cluster.

For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed.`
Defining NDB Cluster Data Nodes

**MaxDiskWriteSpeedOtherNodeRestart**

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations when one or more data nodes in this NDB Cluster are restarting, other than this node.

For setting the maximum rate of disk writes allowed while this data node is restarting, use `MaxDiskWriteSpeedOwnRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

**MaxDiskWriteSpeedOwnRestart**

Set the maximum rate for writing to disk, in bytes per second, by local checkpoints and backup operations while this data node is restarting.

For setting the maximum rate of disk writes allowed while other data nodes are restarting, use `MaxDiskWriteSpeedOtherNodeRestart`. For setting the maximum rate of disk writes allowed when no data nodes are restarting anywhere in the cluster, use `MaxDiskWriteSpeed`. The minimum speed for disk writes by all LCPs and backup operations can be adjusted by setting `MinDiskWriteSpeed`.

**MinDiskWriteSpeed**

Set the minimum rate for writing to disk, in bytes per second, by local checkpoints and backup operations.

The maximum rates of disk writes allowed for LCPs and backups under various conditions are adjustable using the parameters `MaxDiskWriteSpeed`, `MaxDiskWriteSpeedOwnRestart`, and `MaxDiskWriteSpeedOtherNodeRestart`. See the descriptions of these parameters for more information.

**ArbitrationTimeout**

This parameter specifies how long data nodes wait for a response from the arbitrator to an arbitration message. If this is exceeded, the network is assumed to have split.

The default value is 7500 milliseconds (7.5 seconds).

**Arbitration**

The `Arbitration` parameter enables a choice of arbitration schemes, corresponding to one of 3 possible values for this parameter:

- **Default.** This enables arbitration to proceed normally, as determined by the `ArbitrationRank` settings for the management and API nodes. This is the default value.

- **Disabled.** Setting `Arbitration = Disabled` in the `[ndbd default]` section of the `config.ini` file to accomplish the same task as setting `ArbitrationRank` to 0 on all management and API nodes. When `Arbitration` is set in this way, any `ArbitrationRank` settings are ignored.

- **WaitExternal.** The `Arbitration` parameter also makes it possible to configure arbitration in such a way that the cluster waits until after the time determined by `ArbitrationTimeout` has passed for an external cluster manager application to perform arbitration instead of handling arbitration internally. This can be done by setting `Arbitration = WaitExternal` in the `[ndbd default]` section of the `config.ini` file. For best results with the `WaitExternal` setting, it
is recommended that `ArbitrationTimeout` be 2 times as long as the interval required by the external cluster manager to perform arbitration.

**Important**

This parameter should be used only in the `[ndbd default]` section of the cluster configuration file. The behavior of the cluster is unspecified when `Arbitration` is set to different values for individual data nodes.

- **RestartSubscriberConnectTimeout**

  This parameter determines the time that a data node waits for subscribing API nodes to connect. Once this timeout expires, any "missing" API nodes are disconnected from the cluster. To disable this timeout, set `RestartSubscriberConnectTimeout` to 0.

  While this parameter is specified in milliseconds, the timeout itself is resolved to the next-greatest whole second.

**Buffering and logging.** Several `[ndbd]` configuration parameters enable the advanced user to have more control over the resources used by node processes and to adjust various buffer sizes at need.

These buffers are used as front ends to the file system when writing log records to disk. If the node is running in diskless mode, these parameters can be set to their minimum values without penalty due to the fact that disk writes are “faked” by the NDB storage engine’s file system abstraction layer.

- **UndoIndexBuffer**

  The UNDO index buffer, whose size is set by this parameter, is used during local checkpoints. The NDB storage engine uses a recovery scheme based on checkpoint consistency in conjunction with an operational REDO log. To produce a consistent checkpoint without blocking the entire system for writes, UNDO logging is done while performing the local checkpoint. UNDO logging is activated on a single table fragment at a time. This optimization is possible because tables are stored entirely in main memory.

  The UNDO index buffer is used for the updates on the primary key hash index. Inserts and deletes rearrange the hash index; the NDB storage engine writes UNDO log records that map all physical changes to an index page so that they can be undone at system restart. It also logs all active insert operations for each fragment at the start of a local checkpoint.

  Reads and updates set lock bits and update a header in the hash index entry. These changes are handled by the page-writing algorithm to ensure that these operations need no UNDO logging.

  This buffer is 2MB by default. The minimum value is 1MB, which is sufficient for most applications. For applications doing extremely large or numerous inserts and deletes together with large transactions and large primary keys, it may be necessary to increase the size of this buffer. If this buffer is too small, the NDB storage engine issues internal error code 677 (`Index UNDO buffers overloaded`).

  **Important**

  It is not safe to decrease the value of this parameter during a rolling restart.

- **UndoDataBuffer**

  This parameter sets the size of the UNDO data buffer, which performs a function similar to that of the UNDO index buffer, except the UNDO data buffer is used with regard to data memory rather than index memory. This buffer is used during the local checkpoint phase of a fragment for inserts, deletes, and updates.
Because UNDO log entries tend to grow larger as more operations are logged, this buffer is also larger than its index memory counterpart, with a default value of 16MB.

This amount of memory may be unnecessarily large for some applications. In such cases, it is possible to decrease this size to a minimum of 1MB.

It is rarely necessary to increase the size of this buffer. If there is such a need, it is a good idea to check whether the disks can actually handle the load caused by database update activity. A lack of sufficient disk space cannot be overcome by increasing the size of this buffer.

If this buffer is too small and gets congested, the NDB storage engine issues internal error code 891 (Data UNDO buffers overloaded).

---

**Important**

It is not safe to decrease the value of this parameter during a rolling restart.

---

**RedoBuffer**

All update activities also need to be logged. The REDO log makes it possible to replay these updates whenever the system is restarted. The NDB recovery algorithm uses a “fuzzy” checkpoint of the data together with the UNDO log, and then applies the REDO log to play back all changes up to the restoration point.

**RedoBuffer** sets the size of the buffer in which the REDO log is written. The default value is 32MB; the minimum value is 1MB.

If this buffer is too small, the NDB storage engine issues error code 1221 (REDO log buffers overloaded). For this reason, you should exercise care if you attempt to decrease the value of **RedoBuffer** as part of an online change in the cluster’s configuration.

**ndbmtd** allocates a separate buffer for each LDM thread (see **ThreadConfig**). For example, with 4 LDM threads, an **ndbmtd** data node actually has 4 buffers and allocates **RedoBuffer** bytes to each one, for a total of 4 * **RedoBuffer** bytes.

---

**EventLogBufferSize**

Controls the size of the circular buffer used for NDB log events within data nodes.

**Controlling log messages.** In managing the cluster, it is very important to be able to control the number of log messages sent for various event types to **stdout**. For each event category, there are 16 possible event levels (numbered 0 through 15). Setting event reporting for a given event category to level 15 means all event reports in that category are sent to **stdout**; setting it to 0 means that no event reports in that category are made.

By default, only the startup message is sent to **stdout**, with the remaining event reporting level defaults being set to 0. The reason for this is that these messages are also sent to the management server’s cluster log.

An analogous set of levels can be set for the management client to determine which event levels to record in the cluster log.

**LogLevelStartup**

The reporting level for events generated during startup of the process.

The default level is 1.

**LogLevelShutdown**
The reporting level for events generated as part of graceful shutdown of a node.

The default level is 0.

- **LogLevelStatistic**

The reporting level for statistical events such as number of primary key reads, number of updates, number of inserts, information relating to buffer usage, and so on.

The default level is 0.

- **LogLevelCheckpoint**

The reporting level for events generated by local and global checkpoints.

The default level is 0.

- **LogLevelNodeRestart**

The reporting level for events generated during node restart.

The default level is 0.

- **LogLevelConnection**

The reporting level for events generated by connections between cluster nodes.

The default level is 0.

- **LogLevelError**

The reporting level for events generated by errors and warnings by the cluster as a whole. These errors do not cause any node failure but are still considered worth reporting.

The default level is 0.

- **LogLevelCongestion**

The reporting level for events generated by congestion. These errors do not cause node failure but are still considered worth reporting.

The default level is 0.

- **LogLevelInfo**

The reporting level for events generated for information about the general state of the cluster.

The default level is 0.
• **MemReportFrequency**

This parameter controls how often data node memory usage reports are recorded in the cluster log; it is an integer value representing the number of seconds between reports.

Each data node's data memory and index memory usage is logged as both a percentage and a number of 32 KB pages of **DataMemory**, as set in the **config.ini** file. For example, if **DataMemory** is equal to 100 MB, and a given data node is using 50 MB for data memory storage, the corresponding line in the cluster log might look like this:

```
2006-12-24 01:18:16 [MgmSrvr] INFO -- Node 2: Data usage is 50%(1280 32K pages of total 2560)
```

**MemReportFrequency** is not a required parameter. If used, it can be set for all cluster data nodes in the **[ndbd default]** section of **config.ini**, and can also be set or overridden for individual data nodes in the corresponding **[ndbd]** sections of the configuration file. The minimum value—which is also the default value—is 0, in which case memory reports are logged only when memory usage reaches certain percentages (80%, 90%, and 100%), as mentioned in the discussion of statistics events in Section 7.3.2, “NDB Cluster Log Events”.

• **StartupStatusReportFrequency**

When a data node is started with the **--initial** option, it initializes the redo log file during Start Phase 4 (see Section 7.4, “Summary of NDB Cluster Start Phases”). When very large values are set for **NoOfFragmentLogFiles**, **FragmentLogFileSize**, or both, this initialization can take a long time. You can force reports on the progress of this process to be logged periodically, by means of the **StartupStatusReportFrequency** configuration parameter. In this case, progress is reported in the cluster log, in terms of both the number of files and the amount of space that have been initialized, as shown here:

```
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15557
#Total files: 80, Completed: 60
#Total MBytes: 20480, Completed: 15570
```

These reports are logged each **StartupStatusReportFrequency** seconds during Start Phase 4. If **StartupStatusReportFrequency** is 0 (the default), then reports are written to the cluster log only when at the beginning and at the completion of the redo log file initialization process.

**Data Node Debugging Parameters**

The following parameters are intended for use during testing or debugging of data nodes, and not for use in production.

• **DictTrace**

It is possible to cause logging of traces for events generated by creating and dropping tables using **DictTrace**. This parameter is useful only in debugging NDB kernel code. **DictTrace** takes an integer value. 0 is the default, and means no logging is performed; 1 enables trace logging, and 2 enables logging of additional **DBDICT** debugging output.

• **WatchdogImmediateKill**

You can cause threads to be killed immediately whenever watchdog issues occur by enabling the **WatchdogImmediateKill** data node configuration parameter. This parameter should be used only when debugging or troubleshooting, to obtain trace files reporting exactly what was occurring the instant that execution ceased.
Defining NDB Cluster Data Nodes

Backup parameters.  The [ndbd] parameters discussed in this section define memory buffers set aside for execution of online backups.

- **BackupDataBufferSize**

  In creating a backup, there are two buffers used for sending data to the disk. The backup data buffer is used to fill in data recorded by scanning a node's tables. Once this buffer has been filled to the level specified as BackupWriteSize, the pages are sent to disk. While flushing data to disk, the backup process can continue filling this buffer until it runs out of space. When this happens, the backup process pauses the scan and waits until some disk writes have completed freeing up memory so that scanning may continue.

  The default value for this parameter is 16MB. The minimum is 512K.

- **BackupDiskWriteSpeedPct**

  During normal operation, data nodes attempt to maximize the disk write speed used for local checkpoints and backups while remaining within the bounds set by MinDiskWriteSpeed and MaxDiskWriteSpeed. Disk write throttling gives each LDM thread an equal share of the total budget. This allows parallel LCPs to take place without exceeding the disk I/O budget. Because a backup is executed by only one LDM thread, this effectively caused a budget cut, resulting in longer backup completion times, and—if the rate of change is sufficiently high—in failure to complete the backup when the backup log buffer fill rate is higher than the achievable write rate.

  This problem can be addressed by using the BackupDiskWriteSpeedPct configuration parameter, which takes a value in the range 0-90 (inclusive) which is interpreted as the percentage of the node's maximum write rate budget that is reserved prior to sharing out the remainder of the budget among LDM threads for LCPs. The LDM thread running the backup receives the whole write rate budget for the backup, plus its (reduced) share of the write rate budget for local checkpoints.

  The default value for this parameter is 50 (interpreted as 50%).

- **BackupLogBufferSize**

  The backup log buffer fulfills a role similar to that played by the backup data buffer, except that it is used for generating a log of all table writes made during execution of the backup. The same principles apply for writing these pages as with the backup data buffer, except that when there is no more space in the backup log buffer, the backup fails. For that reason, the size of the backup log buffer must be large enough to handle the load caused by write activities while the backup is being made. See Section 7.8.3, "Configuration for NDB Cluster Backups".

  The default value for this parameter should be sufficient for most applications. In fact, it is more likely for a backup failure to be caused by insufficient disk write speed than it is for the backup log buffer to become full. If the disk subsystem is not configured for the write load caused by applications, the cluster is unlikely to be able to perform the desired operations.

  It is preferable to configure cluster nodes in such a manner that the processor becomes the bottleneck rather than the disks or the network connections.

  The default value for this parameter is 16MB.

- **BackupMemory**

  This parameter is deprecated, and subject to removal in a future version of NDB Cluster. Any setting made for it is ignored.

- **BackupReportFrequency**
Defining NDB Cluster Data Nodes

This parameter controls how often backup status reports are issued in the management client during a backup, as well as how often such reports are written to the cluster log (provided cluster event logging is configured to permit it—see Logging and checkpointing). BackupReportFrequency represents the time in seconds between backup status reports.

The default value is 0.

- **BackupWriteSize**

  This parameter specifies the default size of messages written to disk by the backup log and backup data buffers.

  The default value for this parameter is 256KB.

- **BackupMaxWriteSize**

  This parameter specifies the maximum size of messages written to disk by the backup log and backup data buffers.

  The default value for this parameter is 1MB.

- **CompressedBackup**

  Enabling this parameter causes backup files to be compressed. The compression used is equivalent to `gzip --fast`, and can save 50% or more of the space required on the data node to store uncompressed backup files. Compressed backups can be enabled for individual data nodes, or for all data nodes (by setting this parameter in the `[ndbd default]` section of the `config.ini` file).

    **Important**

    You cannot restore a compressed backup to a cluster running a MySQL version that does not support this feature.

  The default value is 0 (disabled).

- **RequireEncryptedBackup**

  If set to 1, backups must be encrypted. While it is possible to set this parameter for each data node individually, it is recommended that you set it in the `[ndbd default]` section of the `config.ini` global configuration file. For more information about performing encrypted backups, see Section 7.8.2, “Using The NDB Cluster Management Client to Create a Backup”.

  Added in NDB 8.0.22.

    **Note**

    The location of the backup files is determined by the `BackupDataDir` data node configuration parameter.

**Additional requirements.** When specifying these parameters, the following relationships must hold true. Otherwise, the data node cannot start.

- **BackupDataBufferSize >= BackupWriteSize + 188KB**
- **BackupLogBufferSize >= BackupWriteSize + 16KB**
- **BackupMaxWriteSize >= BackupWriteSize**
NDB Cluster Realtime Performance Parameters

The \{ndbd\} parameters discussed in this section are used in scheduling and locking of threads to specific CPUs on multiprocessor data node hosts.

**Note**
To make use of these parameters, the data node process must be run as system root.

- **BuildIndexThreads**

  This parameter determines the number of threads to create when rebuilding ordered indexes during a system or node start, as well as when running `ndb_restore --rebuild-indexes`. It is supported only when there is more than one fragment for the table per data node (for example, when `COMMENT="NDB_TABLE=PARTITION_BALANCE=FOR_RA_BY_LDM_X_2"` is used with `CREATE TABLE`).

  Setting this parameter to 0 (the default) disables multithreaded building of ordered indexes.

  This parameter is supported when using `ndbd` or `ndbmtd`.

  You can enable multithreaded builds during data node initial restarts by setting the `TwoPassInitialNodeRestartCopy` data node configuration parameter to `TRUE`.

- **LockExecuteThreadToCPU**

  When used with `ndbd`, this parameter (now a string) specifies the ID of the CPU assigned to handle the \{NDBCLUSTER\} execution thread. When used with `ndbmtd`, the value of this parameter is a comma-separated list of CPU IDs assigned to handle execution threads. Each CPU ID in the list should be an integer in the range 0 to 65535 (inclusive).

  The number of IDs specified should match the number of execution threads determined by `MaxNoOfExecutionThreads`. However, there is no guarantee that threads are assigned to CPUs in any given order when using this parameter. You can obtain more finely-grained control of this type using `ThreadConfig`.

  **LockExecuteThreadToCPU** has no default value.

- **LockMaintThreadsToCPU**

  This parameter specifies the ID of the CPU assigned to handle \{NDBCLUSTER\} maintenance threads.

  The value of this parameter is an integer in the range 0 to 65535 (inclusive). **There is no default value.**

- **Numa**

  This parameter determines whether Non-Uniform Memory Access (NUMA) is controlled by the operating system or by the data node process, whether the data node uses `ndbd` or `ndbmtd`. By default, NDB attempts to use an interleaved NUMA memory allocation policy on any data node where the host operating system provides NUMA support.

  Setting `Numa = 0` means that the datanode process does not itself attempt to set a policy for memory allocation, and permits this behavior to be determined by the operating system, which may be further guided by the separate `numactl` tool. That is, `Numa = 0` yields the system default behavior, which can be customised by `numactl`. For many Linux systems, the system default behavior is to allocate socket-local memory to any given process at allocation time. This can be
Defining NDB Cluster Data Nodes

prolong problem when using `ndbmtd`; this is because `ndbmtd` allocates all memory at startup, leading to an imbalance, giving different access speeds for different sockets, especially when locking pages in main memory.

Setting `Numa = 1` means that the data node process uses `libnuma` to request interleaved memory allocation. (This can also be accomplished manually, on the operating system level, using `numactl`.) Using interleaved allocation in effect tells the data node process to ignore non-uniform memory access but does not attempt to take any advantage of fast local memory; instead, the data node process tries to avoid imbalances due to slow remote memory. If interleaved allocation is not desired, set `Numa` to 0 so that the desired behavior can be determined on the operating system level.

The `Numa` configuration parameter is supported only on Linux systems where `libnuma.so` is available.

- **RealtimeScheduler**

  Setting this parameter to 1 enables real-time scheduling of data node threads.

  The default is 0 (scheduling disabled).

- **SchedulerExecutionTimer**

  This parameter specifies the time in microseconds for threads to be executed in the scheduler before being sent. Setting it to 0 minimizes the response time; to achieve higher throughput, you can increase the value at the expense of longer response times.

  The default is 50 usec, which our testing shows to increase throughput slightly in high-load cases without materially delaying requests.

- **SchedulerResponsiveness**

  Set the balance in the NDB scheduler between speed and throughput. This parameter takes an integer whose value is in the range 0-10 inclusive, with 5 as the default. Higher values provide better response times relative to throughput. Lower values provide increased throughput at the expense of longer response times.

- **SchedulerSpinTimer**

  This parameter specifies the time in microseconds for threads to be executed in the scheduler before sleeping.

  Starting with NDB 8.0.20, if `SpinMethod` is set, any setting for this parameter is ignored.

- **SpinMethod**

  This parameter is present beginning in NDB 8.0.20, but has no effect prior to NDB 8.0.24. It provides a simple interface to control adaptive spinning on data nodes, with four possible values furnishing presets for spin parameter values, as shown in the following list:

  1. **StaticSpinning** (default): Sets `EnableAdaptiveSpinning` to false and `SchedulerSpinTimer` to 0. (`SetAllowedSpinOverhead` is not relevant in this case.)

  2. **CostBasedSpinning**: Sets `EnableAdaptiveSpinning` to true, `SchedulerSpinTimer` to 100, and `SetAllowedSpinOverhead` to 200.

  3. **LatencyOptimisedSpinning**: Sets `EnableAdaptiveSpinning` to true, `SchedulerSpinTimer` to 200, and `SetAllowedSpinOverhead` to 1000.
4. **DatabaseMachineSpinning**: Sets `EnableAdaptiveSpinning` to `true`, `SchedulerSpinTimer` to `500`, and `SetAllowedSpinOverhead` to `10000`. This is intended for use in cases where threads own their own CPUs.

The spin parameters modified by `SpinMethod` are described in the following list:

- **SchedulerSpinTimer**: This is the same as the data node configuration parameter of that name. The setting applied to this parameter by `SpinMethod` overrides any value set in the `config.ini` file.

- **EnableAdaptiveSpinning**: Enables or disables adaptive spinning. Disabling it causes spinning to be performed without making any checks for CPU resources. This parameter cannot be set directly in the cluster configuration file, and under most circumstances should not need to be, but can be enabled directly using `DUMP 104004 1` or disabled with `DUMP 104004 0` in the `ndb_mgm` management client.

- **SetAllowedSpinOverhead**: Sets the amount of CPU time to allow for gaining latency. This parameter cannot be set directly in the `config.ini` file. In most cases, the setting applied by `SpinMethod` should be satisfactory, but if it is necessary to change it directly, you can use `DUMP 104002 overhead` to do so, where `overhead` is a value ranging from `0` to `10000`, inclusive; see the description of the indicated `DUMP` command for details.

On platforms lacking usable spin instructions, such as PowerPC and some SPARC platforms, spin time is set to `0` in all situations, and values for `SpinMethod` other than `StaticSpinning` are ignored.

- **TwoPassInitialNodeRestartCopy**

Multithreaded building of ordered indexes can be enabled for initial restarts of data nodes by setting this configuration parameter to `true` (the default value), which enables two-pass copying of data during initial node restarts.

You must also set `BuildIndexThreads` to a nonzero value.

**Multi-Threading Configuration Parameters (ndbmt)`ndbmt` runs by default as a single-threaded process and must be configured to use multiple threads, using either of two methods, both of which require setting configuration parameters in the `config.ini` file. The first method is simply to set an appropriate value for the `MaxNoOfExecutionThreads` configuration parameter. A second method, makes it possible to set up more complex rules for `ndbmt` multithreading using `ThreadConfig`. The next few paragraphs provide information about these parameters and their use with multithreaded data nodes.

**Note**

A backup using parallelism on the data nodes requires that multiple LDMs are in use on all data nodes in the cluster prior to taking the backup. For more information, see Section 7.8.5, “Taking an NDB Backup with Parallel Data Nodes”, as well as Section 6.23.3, “Restoring from a backup taken in parallel”.

- **AutomaticThreadConfig**

When set to `1`, enables automatic thread configuration employing the number of CPUs available to a data node taking into account any limits set by `taskset`, `numactl`, virtual machines, Docker, and other such means of controlling which CPUs are available to a given application (on Windows platforms, automatic thread configuration uses all CPUs which are online); alternatively, you can set `NumCPUs` to the desired number of CPUs (up to `1024`, the maximum number of CPUs that can be handled by automatic thread configuration). Any settings for `ThreadConfig` and `MaxNoOfExecutionThreads` are ignored.
Defining NDB Cluster Data Nodes

• **ClassicFragmentation**

When enabled (set to `true`), NDB uses flexible distribution of table fragments among LDMs. This means that the default number of partitions per node is equal to `PartitionsPerNode`, rather than to the minimum number of LDM threads per data node, and that each table has a number of partitions equal to 4 (the default for `PartitionsPerNode`) times the number of data nodes.

For new clusters for which a downgrade to NDB 8.0.22 or earlier is never expected to occur, setting `ClassicFragmentation` to `false` when first setting up the cluster is preferable; this ensures that all partitions are spread out evenly between all LDMs.

• **MaxNoOfExecutionThreads**

This parameter directly controls the number of execution threads used by `ndbmtd`, up to a maximum of 72. Although this parameter is set in `[ndbd]` or `[ndbd default]` sections of the `config.ini` file, it is exclusive to `ndbmtd` and does not apply to `ndbd`.

Setting `MaxNoOfExecutionThreads` sets the number of threads for each type as determined by a matrix in the file `storage/ndb/src/kernel/vm/mt_thr_config.cpp`. This table shows these numbers of threads for possible values of `MaxNoOfExecutionThreads`.

**Table 5.5 MaxNoOfExecutionThreads values and the corresponding number of threads by thread type (LQH, TC, Send, Receive).**

<table>
<thead>
<tr>
<th>MaxNoOfExecutionThreads Value</th>
<th>LDM Threads</th>
<th>TC Threads</th>
<th>Send Threads</th>
<th>Receive Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 .. 3</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>4 .. 6</td>
<td>2</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>7 .. 8</td>
<td>4</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>9</td>
<td>4</td>
<td>2</td>
<td>0</td>
<td>1</td>
</tr>
<tr>
<td>10</td>
<td>4</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
<td>3</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>12</td>
<td>6</td>
<td>2</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>13</td>
<td>6</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>14</td>
<td>6</td>
<td>3</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>15</td>
<td>8</td>
<td>3</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>16</td>
<td>8</td>
<td>4</td>
<td>1</td>
<td>2</td>
</tr>
<tr>
<td>17</td>
<td>8</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>18</td>
<td>8</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>19</td>
<td>10</td>
<td>4</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>20</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>2</td>
</tr>
<tr>
<td>21</td>
<td>10</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>22</td>
<td>10</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>23</td>
<td>10</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>24</td>
<td>12</td>
<td>5</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>25</td>
<td>12</td>
<td>6</td>
<td>2</td>
<td>3</td>
</tr>
<tr>
<td>26</td>
<td>12</td>
<td>6</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>27</td>
<td>12</td>
<td>7</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>28</td>
<td>12</td>
<td>7</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>MaxNoOfExecutionThreads Value</td>
<td>LDM Threads</td>
<td>TC Threads</td>
<td>Send Threads</td>
<td>Receive Threads</td>
</tr>
<tr>
<td>-------------------------------</td>
<td>-------------</td>
<td>------------</td>
<td>--------------</td>
<td>----------------</td>
</tr>
<tr>
<td>29</td>
<td>12</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>30</td>
<td>12</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>31</td>
<td>12</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>32</td>
<td>16</td>
<td>8</td>
<td>3</td>
<td>3</td>
</tr>
<tr>
<td>33</td>
<td>16</td>
<td>8</td>
<td>3</td>
<td>4</td>
</tr>
<tr>
<td>34</td>
<td>16</td>
<td>8</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>35</td>
<td>16</td>
<td>9</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>36</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>37</td>
<td>16</td>
<td>10</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>38</td>
<td>16</td>
<td>11</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>39</td>
<td>16</td>
<td>11</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>40</td>
<td>20</td>
<td>10</td>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>41</td>
<td>20</td>
<td>10</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>42</td>
<td>20</td>
<td>11</td>
<td>4</td>
<td>5</td>
</tr>
<tr>
<td>43</td>
<td>20</td>
<td>11</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>44</td>
<td>20</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>45</td>
<td>20</td>
<td>12</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>46</td>
<td>20</td>
<td>13</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>47</td>
<td>20</td>
<td>13</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>48</td>
<td>24</td>
<td>12</td>
<td>5</td>
<td>5</td>
</tr>
<tr>
<td>49</td>
<td>24</td>
<td>12</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>50</td>
<td>24</td>
<td>13</td>
<td>5</td>
<td>6</td>
</tr>
<tr>
<td>51</td>
<td>24</td>
<td>13</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>52</td>
<td>24</td>
<td>14</td>
<td>6</td>
<td>6</td>
</tr>
<tr>
<td>53</td>
<td>24</td>
<td>14</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>54</td>
<td>24</td>
<td>15</td>
<td>6</td>
<td>7</td>
</tr>
<tr>
<td>55</td>
<td>24</td>
<td>15</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>56</td>
<td>24</td>
<td>16</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>57</td>
<td>24</td>
<td>16</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>58</td>
<td>24</td>
<td>17</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>59</td>
<td>24</td>
<td>17</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>60</td>
<td>24</td>
<td>18</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>61</td>
<td>24</td>
<td>18</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>62</td>
<td>24</td>
<td>19</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>63</td>
<td>24</td>
<td>19</td>
<td>9</td>
<td>9</td>
</tr>
<tr>
<td>64</td>
<td>32</td>
<td>16</td>
<td>7</td>
<td>7</td>
</tr>
<tr>
<td>65</td>
<td>32</td>
<td>16</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>66</td>
<td>32</td>
<td>17</td>
<td>7</td>
<td>8</td>
</tr>
<tr>
<td>67</td>
<td>32</td>
<td>17</td>
<td>8</td>
<td>8</td>
</tr>
<tr>
<td>68</td>
<td>32</td>
<td>18</td>
<td>8</td>
<td>8</td>
</tr>
</tbody>
</table>
Defining NDB Cluster Data Nodes

<table>
<thead>
<tr>
<th>MaxNoOfExecutionThreads Value</th>
<th>LDM Threads</th>
<th>TC Threads</th>
<th>Send Threads</th>
<th>Receive Threads</th>
</tr>
</thead>
<tbody>
<tr>
<td>69</td>
<td>32</td>
<td>18</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>70</td>
<td>32</td>
<td>19</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>71</td>
<td>32</td>
<td>20</td>
<td>8</td>
<td>9</td>
</tr>
<tr>
<td>72</td>
<td>32</td>
<td>20</td>
<td>8</td>
<td>10</td>
</tr>
</tbody>
</table>

There is always one SUMA (replication) thread.

NoOfFragmentLogParts should be set equal to the number of LDM threads used by ndbmtd, as determined by the setting for this parameter. This ratio should not be any greater than 4:1; a configuration in which this is the case is specifically disallowed.

The number of LDM threads also determines the number of partitions used by an NDB table that is not explicitly partitioned; this is the number of LDM threads times the number of data nodes in the cluster. (If ndbd is used on the data nodes rather than ndbmtd, then there is always a single LDM thread; in this case, the number of partitions created automatically is simply equal to the number of data nodes. See Section 3.2, “NDB Cluster Nodes, Node Groups, Fragment Replicas, and Partitions”, for more information.

Adding large tablespaces for Disk Data tables when using more than the default number of LDM threads may cause issues with resource and CPU usage if the disk page buffer is insufficiently large; see the description of the DiskPageBufferMemory configuration parameter, for more information.

The thread types are described later in this section (see ThreadConfig).

Setting this parameter outside the permitted range of values causes the management server to abort on startup with the error Error line number: Illegal value value for parameter MaxNoOfExecutionThreads.

For MaxNoOfExecutionThreads, a value of 0 or 1 is rounded up internally by NDB to 2, so that 2 is considered this parameter’s default and minimum value.

MaxNoOfExecutionThreads is generally intended to be set equal to the number of CPU threads available, and to allocate a number of threads of each type suitable to typical workloads. It does not assign particular threads to specified CPUs. For cases where it is desirable to vary from the settings provided, or to bind threads to CPUs, you should use ThreadConfig instead, which allows you to allocate each thread directly to a desired type, CPU, or both.

The multithreaded data node process always spawns, at a minimum, the threads listed here:

- 1 local query handler (LDM) thread
- 1 receive thread
- 1 subscription manager (SUMA or replication) thread

For a MaxNoOfExecutionThreads value of 8 or less, no TC threads are created, and TC handling is instead performed by the main thread.

Changing the number of LDM threads normally requires a system restart, whether it is changed using this parameter or ThreadConfig, but it is possible to effect the change using a node initial restart (NI) provided the following two conditions are met:

- Each LDM thread handles a maximum of 8 fragments, and
Defining NDB Cluster Data Nodes

- The total number of table fragments is an integer multiple of the number of LDM threads.

  In NDB 8.0, an initial restart is not required to effect a change in this parameter, as it was in some older versions of NDB Cluster.

- `NoOfFragmentLogParts`

  Set the number of log file groups for redo logs belonging to this `ndbmtd`. The value of this parameter should be set equal to the number of LDM threads used by `ndbmtd` as determined by the setting for `MaxNoOfExecutionThreads`. A configuration using more than 4 redo log parts per LDM is disallowed.

  See the description of `MaxNoOfExecutionThreads` for more information.

- `NumCPUs`

  Cause automatic thread configuration to use only this many CPUs. Has no effect if `AutomaticThreadConfig` is not enabled.

- `PartitionsPerNode`

  Sets the number of partitions used on each node when creating a new NDB table. This makes it possible to avoid splitting up tables into an excessive number of partitions when the number of local data managers (LDMs) grows high.

  While it is possible to set this parameter to different values on different data nodes and there are no known issues with doing so, there is also not likely to be any advantage in doing so, and so it is recommended simply to set it once, for all data nodes, in the `[ndbd default]` section of the global `config.ini` file.

- `ThreadConfig`

  This parameter is used with `ndbmtd` to assign threads of different types to different CPUs. Its value is a string whose format has the following syntax:

  ```
  ThreadConfig := entry[,entry[,...]]
  entry := type={param[,param[,...]]}
  type (NDB 8.0.22 and earlier) := ldm | main | recv | send | rep | io | tc | watchdog | idxbld
  type (NDB 8.0.23 and later) := ldm | query | recover | main | recv | send | rep | io | tc | watchdog | idxbld
  param := count=number
    | cpubind=cpu_list
    | cpuset=cpu_list
    | spintime=number
    | realtime={0|1}
    | nosend={0|1}
    | thread_prio={0..10}
    | cpubind_exclusive=cpu_list
    | cpuset_exclusive=cpu_list
  ```

  The curly braces ( {...} ) surrounding the list of parameters are required, even if there is only one parameter in the list.

  A `param` (parameter) specifies any or all of the following information:
  
  - The number of threads of the given type (`count`).
  - The set of CPUs to which the threads of the given type are to be nonexclusively bound. This is determined by either one of `cpubind` or `cpuset`. `cpubind` causes each thread to be bound
Defining NDB Cluster Data Nodes

(nonexclusively) to a CPU in the set; `cpuset` means that each thread is bound (nonexclusively) to the set of CPUs specified.

On Solaris, you can instead specify a set of CPUs to which the threads of the given type are to be bound exclusively. `cpubind_exclusive` causes each thread to be bound exclusively to a CPU in the set; `cpuset_exclusive` means that each thread is bound exclusively to the set of CPUs specified.

Only one of `cpubind`, `cpuset`, `cpubind_exclusive`, or `cpuset_exclusive` can be provided in a single configuration.

• `spintime` determines the wait time in microseconds the thread spins before going to sleep.

The default value for `spintime` is the value of the `SchedulerSpinTimer` data node configuration parameter.

`spintime` does not apply to I/O threads, watchdog, or offline index build threads, and so cannot be set for these thread types.

• `realtime` can be set to 0 or 1. If it is set to 1, the threads run with real-time priority. This also means that `thread_prio` cannot be set.

The `realtime` parameter is set by default to the value of the `RealtimeScheduler` data node configuration parameter.

`realtime` cannot be set for offline index build threads.

• By setting `nosend` to 1, you can prevent a `main`, `ldm`, `rep`, or `tc` thread from assisting the send threads. This parameter is 0 by default, and cannot be used with other types of threads.

• `thread_prio` is a thread priority level that can be set from 0 to 10, with 10 representing the greatest priority. The default is 5. The precise effects of this parameter are platform-specific, and are described later in this section.

The thread priority level cannot be set for offline index build threads.

`thread_prio` settings and effects by platform. The implementation of `thread_prio` differs between Linux/FreeBSD, Solaris, and Windows. In the following list, we discuss its effects on each of these platforms in turn:

• **Linux and FreeBSD**: We map `thread_prio` to a value to be supplied to the `nice` system call. Since a lower niceness value for a process indicates a higher process priority, increasing `thread_prio` has the effect of lowering the `nice` value.

<table>
<thead>
<tr>
<th><code>thread_prio</code> value</th>
<th><code>nice</code> value</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>19</td>
</tr>
<tr>
<td>1</td>
<td>16</td>
</tr>
<tr>
<td>2</td>
<td>12</td>
</tr>
<tr>
<td>3</td>
<td>8</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>-4</td>
</tr>
<tr>
<td>7</td>
<td>-8</td>
</tr>
<tr>
<td>8</td>
<td>-12</td>
</tr>
</tbody>
</table>
### Defining NDB Cluster Data Nodes

<table>
<thead>
<tr>
<th>thread_prio value</th>
<th>nice value</th>
</tr>
</thead>
<tbody>
<tr>
<td>9</td>
<td>-16</td>
</tr>
<tr>
<td>10</td>
<td>-20</td>
</tr>
</tbody>
</table>

Some operating systems may provide for a maximum process niceness level of 20, but this is not supported by all targeted versions; for this reason, we choose 19 as the maximum nice value that can be set.

- **Solaris**: Setting `thread_prio` on Solaris sets the Solaris FX priority, with mappings as shown in the following table:

#### Table 5.7 Mapping of thread_prio to FX priority on Solaris

<table>
<thead>
<tr>
<th>thread_prio value</th>
<th>Solaris FX priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>15</td>
</tr>
<tr>
<td>1</td>
<td>20</td>
</tr>
<tr>
<td>2</td>
<td>25</td>
</tr>
<tr>
<td>3</td>
<td>30</td>
</tr>
<tr>
<td>4</td>
<td>35</td>
</tr>
<tr>
<td>5</td>
<td>40</td>
</tr>
<tr>
<td>6</td>
<td>45</td>
</tr>
<tr>
<td>7</td>
<td>50</td>
</tr>
<tr>
<td>8</td>
<td>55</td>
</tr>
<tr>
<td>9</td>
<td>59</td>
</tr>
<tr>
<td>10</td>
<td>60</td>
</tr>
</tbody>
</table>

A `thread_prio` setting of 9 is mapped on Solaris to the special FX priority value 59, which means that the operating system also attempts to force the thread to run alone on its own CPU core.

- **Windows**: We map `thread_prio` to a Windows thread priority value passed to the Windows API `SetThreadPriority()` function. This mapping is shown in the following table:

#### Table 5.8 Mapping of thread_prio to Windows thread priority

<table>
<thead>
<tr>
<th>thread_prio value</th>
<th>Windows thread priority</th>
</tr>
</thead>
<tbody>
<tr>
<td>0 - 1</td>
<td>THREAD_PRIORITY_LOWEST</td>
</tr>
<tr>
<td>2 - 3</td>
<td>THREAD_PRIORITY_BELOW_NORMAL</td>
</tr>
<tr>
<td>4 - 5</td>
<td>THREAD_PRIORITY_NORMAL</td>
</tr>
<tr>
<td>6 - 7</td>
<td>THREAD_PRIORITY_ABOVE_NORMAL</td>
</tr>
<tr>
<td>8 - 10</td>
<td>THREAD_PRIORITY_HIGHEST</td>
</tr>
</tbody>
</table>

The **type** attribute represents an NDB thread type. The thread types supported, and the range of permitted count values for each, are provided in the following list:

- **ldm**: Local query handler (DBLQH kernel block) that handles data. The more LDM threads that are used, the more highly partitioned the data becomes. (Beginning with NDB 8.0.23, when `ClassicFragmentation` is set to 0, the number of partitions is independent of the number of LDM threads, and depends on the value of `PartitionsPerNode` instead.) Each LDM thread maintains its own sets of data and index partitions, as well as its own redo log. Prior to NDB 8.0.23, the value set for `ldm` must be one of the values 1, 2, 4, 6, 8, 12, 16, 24, or 32. In NDB 8.0.23 and later, it is possible to set `ldm` to any value in the range 1 to 332 inclusive; it also
becomes possible to set it to 0, provided that main, rep, and tc are also 0, and that recv is set to 1; doing this causes ndbmtd to emulate ndbd.

Each LDM thread is normally grouped with 1 query thread to form an LDM group. A set of 4 to 8 LDM groups is grouped into a round robin groups. Each LDM thread can be assisted in execution by any query or threads in the same round robin group. NDB attempts to form round robin groups such that all threads in each round robin group are locked to CPUs that are attached to the same L3 cache, within the limits of the range stated for a round robin group's size.

Changing the number of LDM threads normally requires a system restart to be effective and safe for cluster operations; this requirement is relaxed in certain cases, as explained later in this section. This is also true when this is done using MaxNoOfExecutionThreads.

Adding large tablespaces (hundreds of gigabytes or more) for Disk Data tables when using more than the default number of LDMs may cause issues with resource and CPU usage if DiskPageBufferMemory is not sufficiently large.

- **query** (Added in NDB 8.0.23): A query thread is tied to an LDM and together with it forms an LDM group; acts only on **READ COMMITTED** queries. The number of query threads must be set to 0, 1, 2, or 3 times the number of LDM threads. Query threads are not used, unless this is overridden by setting query to a nonzero value, or by enabling the AutomaticThreadConfig parameter, in which case LDMs behave as they did prior to NDB 8.0.23.

  A query thread also acts as a recovery thread (see next item), although the reverse is not true.

  Changing the number of query threads requires a node restart.

- **recover** (Added in NDB 8.0.23): A recovery thread restores data from a fragment as part of an LCP.

  Changing the number of recovery threads requires a node restart.

- **tc**: Transaction coordinator thread (DBTC kernel block) containing the state of an ongoing transaction. In NDB 8.0.23 and later, the maximum number of TC threads is 128; previously, this was 32.

  Optimally, every new transaction can be assigned to a new TC thread. In most cases 1 TC thread per 2 LDM threads is sufficient to guarantee that this can happen. In cases where the number of writes is relatively small when compared to the number of reads, it is possible that only 1 TC thread per 4 LQH threads is required to maintain transaction states. Conversely, in applications that perform a great many updates, it may be necessary for the ratio of TC threads to LDM threads to approach 1 (for example, 3 TC threads to 4 LDM threads).

  Setting tc to 0 causes TC handling to be done by the main thread. In most cases, this is effectively the same as setting it to 1.

  Range: 0-64 (**NDB 8.0.22 and earlier**: 0 - 32)
Defining NDB Cluster Data Nodes

- **main**: Data dictionary and transaction coordinator (**DBDIH** and **DBTC** kernel blocks), providing schema management. Prior to NDB 8.0.23, this was always handled by a single dedicated thread, beginning with NDB 8.0.23, it is also possible to specify zero or two main threads.

  Range:

  - **NDB 8.0.22 and earlier**: 1 only.
  - **NDB 8.0.23 and later**: 0-2.

  Setting `main` to 0 and `rep` to 1 causes the `main` blocks to be placed into the `rep` thread; the combined thread is shown in the `ndbinfo.threads` table as `main_rep`. This is effectively the same as setting `rep` equal to 1 and `main` equal to 0.

  It is also possible to set both `main` and `rep` to 0, in which case both threads are placed in the first `recv` thread; the resulting combined thread is named `main_rep_recv` in the `threads` table.

- **recv**: Receive thread (**CMVMI** kernel block). Each receive thread handles one or more sockets for communicating with other nodes in an NDB Cluster, with one socket per node. NDB Cluster supports multiple receive threads; the maximum is 16 such threads.

  Range:

  - **NDB 8.0.22 and earlier**: 1 - 16
  - **NDB 8.0.23 and later**: 1 - 64

- **send**: Send thread (**CMVMI** kernel block). To increase throughput, it is possible to perform sends from one or more separate, dedicated threads (maximum 8).

  In NDB 8.0.20 and later, due to changes in the multithreading implementation, using many send threads can have an adverse effect on scalability.

  Previously, all threads handled their own sending directly; this can still be made to happen by setting the number of send threads to 0 (this also happens when `MaxNoOfExecutionThreads` is set less than 10). While doing so can have an adverse impact on throughput, it can also in some cases provide decreased latency.

  Range:

  - **NDB 8.0.22 and earlier**: 0 - 16
  - **NDB 8.0.23 and later**: 0 - 64
Defining NDB Cluster Data Nodes

- **rep**: Replication thread (SUMA kernel block). Prior to NDB 8.0.23, asynchronous replication operations are always handled by a single, dedicated thread. Beginning with NDB 8.0.23, this thread can be combined with the main thread (see range information).

  Range:
  - *NDB 8.0.22 and earlier*: 1 only.
  - *NDB 8.0.23 and later*: 0-1.

  Setting `rep` to 0 and `main` to 1 causes the `rep` blocks to be placed into the `main` thread; the combined thread is shown in the `ndbinfo.threads` table as `main_rep`. This is effectively the same as setting `main` equal to 1 and `rep` equal to 0.

  It is also possible to set both `main` and `rep` to 0, in which case both threads are placed in the first `recv` thread; the resulting combined thread is named `main_rep_recv` in the `threads` table.

- **io**: File system and other miscellaneous operations. These are not demanding tasks, and are always handled as a group by a single, dedicated I/O thread.

  Range: 1 only.

- **watchdog**: Parameters settings associated with this type are actually applied to several threads, each having a specific use. These threads include the `SocketServer` thread, which receives connection setups from other nodes; the `SocketClient` thread, which attempts to set up connections to other nodes; and the thread watchdog thread that checks that threads are progressing.

  Range: 1 only.

- **idxbld**: Offline index build threads. Unlike the other thread types listed previously, which are permanent, these are temporary threads which are created and used only during node or system restarts, or when running `ndb_restore --rebuild-indexes`. They may be bound to CPU sets which overlap with CPU sets bound to permanent thread types.

  `thread_prio`, `realtime`, and `spintime` values cannot be set for offline index build threads. In addition, `count` is ignored for this type of thread.

  If `idxbld` is not specified, the default behavior is as follows:

  - Offline index build threads are not bound if the I/O thread is also not bound, and these threads use any available cores.
  - If the I/O thread is bound, then the offline index build threads are bound to the entire set of bound threads, due to the fact that there should be no other tasks for these threads to perform.

  Range: 0 - 1.

Changing `ThreadConfig` normally requires a system initial restart, but this requirement can be relaxed under certain circumstances:

- If, following the change, the number of LDM threads remains the same as before, nothing more than a simple node restart (rolling restart, or `N`) is required to implement the change.
Defining NDB Cluster Data Nodes

- Otherwise (that is, if the number of LDM threads changes), it is still possible to effect the change using a node initial restart (NI) provided the following two conditions are met:
  
  a. Each LDM thread handles a maximum of 8 fragments, and

  b. The total number of table fragments is an integer multiple of the number of LDM threads.

In any other case, a system initial restart is needed to change this parameter.

NDB can distinguish between thread types by both of the following criteria:

- Whether the thread is an execution thread. Threads of type main, ldm, query (NDB 8.0.23 and later), recv, rep, tc, and send are execution threads; io, recover (NDB 8.0.23 and later), watchdog, and idxbld threads are not considered execution threads.

- Whether the allocation of threads to a given task is permanent or temporary. Currently all thread types except idxbld are considered permanent; idxbld threads are regarded as temporary threads.

Simple examples:

```plaintext
# Example 1.
ThreadConfig=ldm={count=2,cpubind=1,2},main={cpubind=12},rep={cpubind=11}
# Example 2.
Threadconfig=main={cpubind=0},ldm={count=4,cpubind=1,2,5,6},io={cpubind=3}
```

It is usually desirable when configuring thread usage for a data node host to reserve one or more number of CPUs for operating system and other tasks. Thus, for a host machine with 24 CPUs, you might want to use 20 CPU threads (leaving 4 for other uses), with 8 LDM threads, 4 TC threads (half the number of LDM threads), 3 send threads, 3 receive threads, and 1 thread each for schema management, asynchronous replication, and I/O operations. (This is almost the same distribution of threads used when MaxNoOfExecutionThreads is set equal to 20.) The following ThreadConfig setting performs these assignments, additionally binding all of these threads to specific CPUs:

```plaintext
ThreadConfig=ldm{count=8,cpubind=1,2,3,4,5,6,7,8},main={cpubind=9},io={cpubind=9}, 
rep={cpubind=10},tc{count=4,cpubind=11,12,13,14},recv={count=3,cpubind=15,16,17}, 
send{count=3,cpubind=18,19,20}
```

It should be possible in most cases to bind the main (schema management) thread and the I/O thread to the same CPU, as we have done in the example just shown.

The following example incorporates groups of CPUs defined using both cpuset and cpubind, as well as use of thread prioritization.

```plaintext
ThreadConfig=ldm{count=4,cpuset=0-3,thread_prio=8,spintime=200}, 
 ldm{count=4,cpubind=4-7,thread_prio=8,spintime=200}, 
 tc{count=4,cpuset=8-9,thread_prio=6},send={count=2,thread_prio=10,cpubind=10-11}, 
 main={count=1,cpubind=10},rep={count=1,cpubind=11}
```

In this case we create two LDM groups; the first uses cpubind and the second uses cpuset. thread_prio and spintime are set to the same values for each group. This means there are eight LDM threads in total. (You should ensure that NoOffFragmentLogParts is also set to 8.) The four TC threads use only two CPUs; it is possible when using cpuset to specify fewer CPUs than threads in the group. (This is not true for cpubind.) The send threads use two threads using cpubind to bind these threads to CPUs 10 and 11. The main and rep threads can reuse these CPUs.

This example shows how ThreadConfig and NoOffFragmentLogParts might be set up for a 24-CPU host with hyperthreading, leaving CPUs 10, 11, 22, and 23 available for operating system functions and interrupts:

```plaintext
NoOffFragmentLogParts=10
```
Defining NDB Cluster Data Nodes

ThreadConfig=ldm={count=10,cpubind=0-4,12-16,thread_prio=9,spintime=200},
  tc={count=4,cpuset=6-7,18-19,thread_prio=8},send={count=1,cpuset=8},
  recv={count=1,cpuset=20},main={count=1,cpuset=9,21},rep={count=1,cpuset=9,21},
  io={count=1,cpuset=9,21,thread_prio=8},watchdog={count=1,cpuset=9,21,thread_prio=9}

The next few examples include settings for idxbld. The first two of these demonstrate how a CPU set defined for idxbld can overlap those specified for other (permanent) thread types, the first using cpuset and the second using cpubind:

ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},
  io={cpubind=8},idxbld={cpuset=1-8}
ThreadConfig=main,ldm={count=1,cpubind=1},idxbld={count=1,cpubind=1}

The next example specifies a CPU for the I/O thread, but not for the index build threads:

ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},
  io={cpubind=8}

Since the ThreadConfig setting just shown locks threads to eight cores numbered 1 through 8, it is equivalent to the setting shown here:

ThreadConfig=main,ldm={count=4,cpuset=1-4},tc={count=4,cpuset=5,6,7},
  io={cpubind=8},idxbld={cpuset=1,2,3,4,5,6,7,8}

In order to take advantage of the enhanced stability that the use of ThreadConfig offers, it is necessary to ensure that CPUs are isolated, and that they are not subject to interrupts, or to being scheduled for other tasks by the operating system. On many Linux systems, you can do this by setting IRQBALANCE_BANNED_CPUS in /etc/sysconfig/irqbalance to 0xFFFFF0, and by using the isolcpus boot option in grub.conf. For specific information, see your operating system or platform documentation.

Disk Data Configuration Parameters. Configuration parameters affecting Disk Data behavior include the following:

- **DiskPageBufferEntries**

  This is the number of page entries (page references) to allocate. It is specified as a number of 32K pages in DiskPageBufferMemory. The default is sufficient for most cases but you may need to increase the value of this parameter if you encounter problems with very large transactions on Disk Data tables. Each page entry requires approximately 100 bytes.

- **DiskPageBufferMemory**

  This determines the amount of space used for caching pages on disk, and is set in the [ndbd] or [ndbd default] section of the config.ini file.

  **Note**

  Previously, this parameter was specified as a number of 32 KB pages. Beginning with NDB 8.0.19, it is specified as a number of bytes.

  If the value for DiskPageBufferMemory is set too low in conjunction with using more than the default number of LDM threads in ThreadConfig (for example {ldm=6...}), problems can arise when trying to add a large (for example 500G) data file to a disk-based NDB table, wherein the process takes indefinitely long while occupying one of the CPU cores.

  This is due to the fact that, as part of adding a data file to a tablespace, extent pages are locked into memory in an extra PGMAN worker thread, for quick metadata access. When adding a large file, this worker has insufficient memory for all of the data file metadata. In such cases, you should either increase DiskPageBufferMemory, or add smaller tablespace files. You may also need to adjust DiskPageBufferEntries.
You can query the `ndbinfodiskpagebuffer` table to help determine whether the value for this parameter should be increased to minimize unnecessary disk seeks. See Section 7.14.27, “The `ndbinfodiskpagebuffer` Table”, for more information.

- **SharedGlobalMemory**

  This parameter determines the amount of memory that is used for log buffers, disk operations (such as page requests and wait queues), and metadata for tablespaces, log file groups, **UNDO** files, and data files. The shared global memory pool also provides memory used for satisfying the memory requirements of the `UNDO_BUFFER_SIZE` option used with `CREATE LOGFILE GROUP` and `ALTER LOGFILE GROUP` statements, including any default value implied for this options by the setting of the `InitialLogFileGroup` data node configuration parameter. `SharedGlobalMemory` can be set in the `[ndbd]` or `[ndbd default]` section of the `config.ini` configuration file, and is measured in bytes.

  The default value is **128M**.

- **DiskIOThreadPool**

  This parameter determines the number of unbound threads used for Disk Data file access. Before `DiskIOThreadPool` was introduced, exactly one thread was spawned for each Disk Data file, which could lead to performance issues, particularly when using very large data files. With `DiskIOThreadPool`, you can—for example—access a single large data file using several threads working in parallel.

  This parameter applies to Disk Data I/O threads only.

  The optimum value for this parameter depends on your hardware and configuration, and includes these factors:

  - **Physical distribution of Disk Data files.** You can obtain better performance by placing data files, undo log files, and the data node file system on separate physical disks. If you do this with some or all of these sets of files, then you can (and should) set `DiskIOThreadPool` higher to enable separate threads to handle the files on each disk.

    In NDB 8.0.19 and later, you should also disable `DiskDataUsingSameDisk` when using a separate disk or disks for Disk Data files; this increases the rate at which checkpoints of Disk Data tablespaces can be performed.

  - **Disk performance and types.** The number of threads that can be accommodated for Disk Data file handling is also dependent on the speed and throughput of the disks. Faster disks and higher throughput allow for more disk I/O threads. Our test results indicate that solid-state disk drives can handle many more disk I/O threads than conventional disks, and thus higher values for `DiskIOThreadPool`.

    Decreasing `TimeBetweenGlobalCheckpoints` is also recommended when using solid-state disk drives, in particular those using NVMe. See also **Disk Data latency parameters**.

  The default value for this parameter is 2.

- **Disk Data file system parameters.** The parameters in the following list make it possible to place NDB Cluster Disk Data files in specific directories without the need for using symbolic links.

  - **FileSystemPathDD**

    If this parameter is specified, then NDB Cluster Disk Data data files and undo log files are placed in the indicated directory. This can be overridden for data files, undo log files, or both, by
Defining NDB Cluster Data Nodes

specifying values for `FileSystemPathDataFiles`, `FileSystemPathUndoFiles`, or both, as explained for these parameters. It can also be overridden for data files by specifying a path in the `ADD DATAFILE` clause of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement, and for undo log files by specifying a path in the `ADD UNDOFILE` clause of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement. If `FileSystemPathDD` is not specified, then `FileSystemPath` is used.

If a `FileSystemPathDD` directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd_defaut]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

• `FileSystemPathDataFiles` If this parameter is specified, then NDB Cluster Disk Data data files are placed in the indicated directory. This overrides any value set for `FileSystemPathDD`. This parameter can be overridden for a given data file by specifying a path in the `ADD DATAFILE` clause of a `CREATE TABLESPACE` or `ALTER TABLESPACE` statement used to create that data file. If `FileSystemPathDataFiles` is not specified, then `FileSystemPathDD` is used (or `FileSystemPath`, if `FileSystemPathDD` has also not been set).

If a `FileSystemPathDataFiles` directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd default]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

• `FileSystemPathUndoFiles` If this parameter is specified, then NDB Cluster Disk Data undo log files are placed in the indicated directory. This overrides any value set for `FileSystemPathDD`. This parameter can be overridden for a given data file by specifying a path in the `ADD UNDOFILE` clause of a `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP` statement used to create that data file. If `FileSystemPathUndoFiles` is not specified, then `FileSystemPathDD` is used (or `FileSystemPath`, if `FileSystemPathDD` has also not been set).

If a `FileSystemPathUndoFiles` directory is specified for a given data node (including the case where the parameter is specified in the `[ndbd default]` section of the `config.ini` file), then starting that data node with `--initial` causes all files in the directory to be deleted.

For more information, see Section 7.10.1, “NDB Cluster Disk Data Objects”.

• Disk Data object creation parameters. The next two parameters enable you—when starting the cluster for the first time—to cause a Disk Data log file group, tablespace, or both, to be created without the use of SQL statements.

• `InitialLogFileGroup` This parameter can be used to specify a log file group that is created when performing an initial start of the cluster. `InitialLogFileGroup` is specified as shown here:

```
InitialLogFileGroup = [name=name;] [undo_buffer_size=size;] file-specification-list
```

The `name` of the log file group is optional and defaults to `DEFAULT-LG`. The `undo_buffer_size` is also optional; if omitted, it defaults to `64M`. Each `file-specification` corresponds to an undo log file, and at least one must be specified in the `file-specification-list`. Undo log files are placed according to any values that have been set for `FileSystemPath`,
Defining NDB Cluster Data Nodes

FilesystemPathDD, and FilesystemPathUndoFiles, just as if they had been created as the result of a CREATE LOGFILE GROUP or ALTER LOGFILE GROUP statement.

Consider the following:

InitialLogfileGroup = name=LG1; undo_buffer_size=128M; undo1.log:250M; undo2.log:150M

This is equivalent to the following SQL statements:

CREATE LOGFILE GROUP LG1
  ADD UNDOFILE 'undo1.log'
  INITIAL_SIZE 250M
  UNDO_BUFFER_SIZE 128M
  ENGINE NDDBCLUSTER;
ALTER LOGFILE GROUP LG1
  ADD UNDOFILE 'undo2.log'
  INITIAL_SIZE 150M
  ENGINE NDDBCLUSTER;

This logfile group is created when the data nodes are started with --initial.

Resources for the initial log file group are added to the global memory pool along with those indicated by the value of SharedGlobalMemory.

This parameter, if used, should always be set in the [ndbd default] section of the config.ini file. The behavior of an NDB Cluster when different values are set on different data nodes is not defined.

- InitialTablespace

This parameter can be used to specify an NDB Cluster Disk Data tablespace that is created when performing an initial start of the cluster. InitialTablespace is specified as shown here:

InitialTablespace = [name=name]; [extent_size=size]; file-specification-list

The name of the tablespace is optional and defaults to DEFAULT-TS. The extent_size is also optional; it defaults to 1M. The file-specification-list uses the same syntax as shown with the InitialLogfileGroup parameter, the only difference being that each file-specification used with InitialTablespace corresponds to a data file. At least one must be specified in the file-specification-list. Data files are placed according to any values that have been set for FilesystemPath, FilesystemPathDD, and FilesystemPathDataFiles, just as if they had been created as the result of a CREATE TABLESPACE or ALTER TABLESPACE statement.

For example, consider the following line specifying InitialTablespace in the [ndbd default] section of the config.ini file (as with InitialLogfileGroup, this parameter should always be set in the [ndbd default] section, as the behavior of an NDB Cluster when different values are set on different data nodes is not defined):

InitialTablespace = name=TS1; extent_size=8M; data1.dat:2G; data2.dat:4G

This is equivalent to the following SQL statements:

CREATE TABLESPACE TS1
  ADD DATAFILE 'data1.dat'
  EXTENT_SIZE 8M
  INITIAL_SIZE 2G
  ENGINE NDDBCLUSTER;
ALTER TABLESPACE TS1
  ADD DATAFILE 'data2.dat'
  INITIAL_SIZE 4G
This tablespace is created when the data nodes are started with `--initial`, and can be used whenever creating NDB Cluster Disk Data tables thereafter.

- **Disk Data latency parameters.** The two parameters listed here can be used to improve handling of latency issues with NDB Cluster Disk Data tables.
  - **MaxDiskDataLatency**
    
    This parameter controls the maximum allowed mean latency for disk access (maximum 8000 milliseconds). When this limit is reached, NDB begins to abort transactions in order to decrease pressure on the Disk Data I/O subsystem. Use 0 to disable the latency check.
  - **DiskDataUsingSameDisk**
    
    Set this parameter to false if your Disk Data tablespaces use one or more separate disks. Doing so allows checkpoints to tablespaces to be executed at a higher rate than normally used for when disks are shared.

    When `DiskDataUsingSameDisk` is true, NDB decreases the rate of Disk Data checkpointing whenever an in-memory checkpoint is in progress to help ensure that disk load remains constant.

**Disk Data and GCP Stop errors.** Errors encountered when using Disk Data tables such as `Node nodeid killed this node because GCP stop was detected` (error 2303) are often referred to as “GCP stop errors”. Such errors occur when the redo log is not flushed to disk quickly enough; this is usually due to slow disks and insufficient disk throughput.

You can help prevent these errors from occurring by using faster disks, and by placing Disk Data files on a separate disk from the data node file system. Reducing the value of `TimeBetweenGlobalCheckpoints` tends to decrease the amount of data to be written for each global checkpoint, and so may provide some protection against redo log buffer overflows when trying to write a global checkpoint; however, reducing this value also permits less time in which to write the GCP, so this must be done with caution.

In addition to the considerations given for `DiskPageBufferMemory` as explained previously, it is also very important that the `DiskIOThreadPool` configuration parameter be set correctly; having `DiskIOThreadPool` set too high is very likely to cause GCP stop errors (Bug #37227).

GCP stops can be caused by save or commit timeouts; the `TimeBetweenEpochsTimeout` data node configuration parameter determines the timeout for commits. However, it is possible to disable both types of timeouts by setting this parameter to 0.

**Parameters for configuring send buffer memory allocation.** Send buffer memory is allocated dynamically from a memory pool shared between all transporters, which means that the size of the send buffer can be adjusted as necessary. (Previously, the NDB kernel used a fixed-size send buffer for every node in the cluster, which was allocated when the node started and could not be changed while the node was running.) The `TotalSendBufferMemory` and `OverLoadLimit` data node configuration parameters permit the setting of limits on this memory allocation. For more information about the use of these parameters (as well as `SendBufferMemory`), see Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

- **ExtraSendBufferMemory**
  
  This parameter specifies the amount of transporter send buffer memory to allocate in addition to any set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- **TotalSendBufferMemory**
Defining NDB Cluster Data Nodes

This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

See also Section 7.7, “Adding NDB Cluster Data Nodes Online”.

Redo log over-commit handling. It is possible to control a data node’s handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than \( \text{RedoOverCommitLimit} \) seconds, more than \( \text{RedoOverCommitCounter} \) times, causing any pending transactions to be aborted. When this happens, the API node that sent the transaction can handle the operations that should have been committed either by queuing the operations and re-trying them, or by aborting them, as determined by \( \text{DefaultOperationRedoProblemAction} \). The data node configuration parameters for setting the timeout and number of times it may be exceeded before the API node takes this action are described in the following list:

- \( \text{RedoOverCommitCounter} \)

  When \( \text{RedoOverCommitLimit} \) is exceeded when trying to write a given redo log to disk this many times or more, any transactions that were not committed as a result are aborted, and an API node where any of these transactions originated handles the operations making up those transactions according to its value for \( \text{DefaultOperationRedoProblemAction} \) (by either queuing the operations to be re-trying, or aborting them).

- \( \text{RedoOverCommitLimit} \)

  This parameter sets an upper limit in seconds for trying to write a given redo log to disk before timing out. The number of times the data node tries to flush this redo log, but takes longer than \( \text{RedoOverCommitLimit} \), is kept and compared with \( \text{RedoOverCommitCounter} \), and when flushing takes too long more times than the value of that parameter, any transactions that were not committed as a result of the flush timeout are aborted. When this occurs, the API node where any of these transactions originated handles the operations making up those transactions according to its \( \text{DefaultOperationRedoProblemAction} \) setting (it either queues the operations to be re-trying, or aborts them).

Controlling restart attempts. It is possible to exercise finely-grained control over restart attempts by data nodes when they fail to start using the \( \text{MaxStartFailRetries} \) and \( \text{StartFailRetryDelay} \) data node configuration parameters.

\( \text{MaxStartFailRetries} \) limits the total number of retries made before giving up on starting the data node, \( \text{StartFailRetryDelay} \) sets the number of seconds between retry attempts. These parameters are listed here:

- \( \text{StartFailRetryDelay} \)

  Use this parameter to set the number of seconds between restart attempts by the data node in the event on failure on startup. The default is 0 (no delay).

  Both this parameter and \( \text{MaxStartFailRetries} \) are ignored unless \( \text{StopOnError} \) is equal to 0.

- \( \text{MaxStartFailRetries} \)

  Use this parameter to limit the number restart attempts made by the data node in the event that it fails on startup. The default is 3 attempts.
Both this parameter and \texttt{StartFailRetryDelay} are ignored unless \texttt{StopOnError} is equal to 0.

\textbf{NDB index statistics parameters.} The parameters in the following list relate to NDB index statistics generation.

- \texttt{IndexStatAutoCreate}

Enable (set equal to 1) or disable (set equal to 0) automatic statistics collection when indexes are created. Disabled by default.

- \texttt{IndexStatAutoUpdate}

Enable (set equal to 1) or disable (set equal to 0) monitoring of indexes for changes and trigger automatic statistics updates these are detected. The amount and degree of change needed to trigger the updates are determined by the settings for the \texttt{IndexStatTriggerPct} and \texttt{IndexStatTriggerScale} options.

- \texttt{IndexStatSaveSize}

Maximum space in bytes allowed for the saved statistics of any given index in the NDB system tables and in the \texttt{mysqld} memory cache.

At least one sample is always produced, regardless of any size limit. This size is scaled by \texttt{IndexStatSaveScale}.

The size specified by \texttt{IndexStatSaveSize} is scaled by the value of \texttt{IndexStatTriggerPct} for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting \texttt{IndexStatTriggerPct} equal to 0 disables the scaling effect.

- \texttt{IndexStatSaveScale}

The size specified by \texttt{IndexStatSaveSize} is scaled by the value of \texttt{IndexStatTriggerPct} for a large index, times 0.01. This is further multiplied by the logarithm to the base 2 of the index size. Setting \texttt{IndexStatTriggerPct} equal to 0 disables the scaling effect.

- \texttt{IndexStatTriggerPct}

Percentage change in updates that triggers an index statistics update. The value is scaled by \texttt{IndexStatTriggerScale}. You can disable this trigger altogether by setting \texttt{IndexStatTriggerPct} to 0.

- \texttt{IndexStatTriggerScale}

Scale \texttt{IndexStatTriggerPct} by this amount times 0.01 for a large index. A value of 0 disables scaling.

- \texttt{IndexStatUpdateDelay}

Minimum delay in seconds between automatic index statistics updates for a given index. Setting this variable to 0 disables any delay. The default is 60 seconds.

\textbf{Restart types.} Information about the restart types used by the parameter descriptions in this section is shown in the following table:
5.3.7 Defining SQL and Other API Nodes in an NDB Cluster

The [mysqld] and [api] sections in the config.ini file define the behavior of the MySQL servers (SQL nodes) and other applications (API nodes) used to access cluster data. None of the parameters shown is required. If no computer or host name is provided, any host can use this SQL or API node.

Generally speaking, a [mysqld] section is used to indicate a MySQL server providing an SQL interface to the cluster, and an [api] section is used for applications other than mysqld processes accessing cluster data, but the two designations are actually synonymous; you can, for instance, list parameters for a MySQL server acting as an SQL node in an [api] section.

Note

For a discussion of MySQL server options for NDB Cluster, see Section 5.3.9.1, “MySQL Server Options for NDB Cluster”. For information about MySQL server system variables relating to NDB Cluster, see Section 5.3.9.2, “NDB Cluster System Variables”.

• Id

The Id is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.

Note

In NDB 8.0.18 and later, data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 144. (Previous to NDB 8.0.18, the maximum supported value for a data node ID was 48.)

NodeId is the preferred parameter name to use when identifying API nodes. (id continues to be supported for backward compatibility, but is now deprecated and generates a warning when used. It is also subject to future removal.)

• ConnectionMap

Specifies which data nodes to connect.

• NodeId

The NodeId is an integer value used to identify the node in all cluster internal messages. The permitted range of values is 1 to 255 inclusive. This value must be unique for each node in the cluster, regardless of the type of node.
Defining SQL and Other API Nodes in an NDB Cluster

Note

In NDB 8.0.18 and later, data node IDs must be less than 145. If you plan to deploy a large number of data nodes, it is a good idea to limit the node IDs for API nodes (and management nodes) to values greater than 144. (Previous to NDB 8.0.18, the maximum supported value for a data node ID was 48.)

NodeId is the preferred parameter name to use when identifying management nodes. An alias, Id, was used for this purpose in very old versions of NDB Cluster, and continues to be supported for backward compatibility; it is now deprecated and generates a warning when used, and is subject to removal in a future release of NDB Cluster.

- **ExecuteOnComputer**

  This refers to the Id set for one of the computers (hosts) defined in a [computer] section of the configuration file.

  **Important**

  This parameter is deprecated, and is subject to removal in a future release. Use the HostName parameter instead.

- **HostName**

  Specifying this parameter defines the hostname of the computer on which the SQL node (API node) is to reside. To specify a hostname, either this parameter or ExecuteOnComputer is required.

  If no HostName or ExecuteOnComputer is specified in a given [mysql] or [api] section of the config.ini file, then an SQL or API node may connect using the corresponding “slot” from any host which can establish a network connection to the management server host machine. This differs from the default behavior for data nodes, where localhost is assumed for HostName unless otherwise specified.

- **LocationDomainId**

  Assigns an SQL or other API node to a specific availability domain (also known as an availability zone) within a cloud. By informing NDB which nodes are in which availability domains, performance can be improved in a cloud environment in the following ways:

  - If requested data is not found on the same node, reads can be directed to another node in the same availability domain.
  
  - Communication between nodes in different availability domains are guaranteed to use NDB transporters’ WAN support without any further manual intervention.
  
  - The transporter’s group number can be based on which availability domain is used, such that also SQL and other API nodes communicate with local data nodes in the same availability domain whenever possible.
  
  - The arbitrator can be selected from an availability domain in which no data nodes are present, or, if no such availability domain can be found, from a third availability domain.
**LocationDomainId** takes an integer value between 0 and 16 inclusive, with 0 being the default; using 0 is the same as leaving the parameter unset.

- **ArbitrationRank**

  This parameter defines which nodes can act as arbitrators. Both management nodes and SQL nodes can be arbitrators. A value of 0 means that the given node is never used as an arbitrator, a value of 1 gives the node high priority as an arbitrator, and a value of 2 gives it low priority. A normal configuration uses the management server as arbitrator, setting its `ArbitrationRank` to 1 (the default for management nodes) and those for all SQL nodes to 0 (the default for SQL nodes).

  By setting `ArbitrationRank` to 0 on all management and SQL nodes, you can disable arbitration completely. You can also control arbitration by overriding this parameter; to do so, set the `Arbitration` parameter in the `[ndbd default]` section of the `config.ini` global configuration file.

- **ArbitrationDelay**

  Setting this parameter to any other value than 0 (the default) means that responses by the arbitrator to arbitration requests are delayed by the stated number of milliseconds. It is usually not necessary to change this value.

- **BatchByteSize**

  For queries that are translated into full table scans or range scans on indexes, it is important for best performance to fetch records in properly sized batches. It is possible to set the proper size both in terms of number of records (`BatchSize`) and in terms of bytes (`BatchByteSize`). The actual batch size is limited by both parameters.

  The speed at which queries are performed can vary by more than 40% depending upon how this parameter is set.

  This parameter is measured in bytes. The default value is 16K.

- **BatchSize**

  This parameter is measured in number of records and is by default set to 256. The maximum size is 992.

- **ExtraSendBufferMemory**

  This parameter specifies the amount of transporter send buffer memory to allocate in addition to any that has been set using `TotalSendBufferMemory`, `SendBufferMemory`, or both.

- **HeartbeatThreadPriority**

  Use this parameter to set the scheduling policy and priority of heartbeat threads for management and API nodes. The syntax for setting this parameter is shown here:

  ```
  HeartbeatThreadPriority = policy[, priority]
  policy:     \{FIFO | RR\}
  ```

  When setting this parameter, you must specify a policy. This is one of FIFO (first in, first in) or RR (round robin). This followed optionally by the priority (an integer).

- **MaxScanBatchSize**
The batch size is the size of each batch sent from each data node. Most scans are performed in parallel to protect the MySQL Server from receiving too much data from many nodes in parallel; this parameter sets a limit to the total batch size over all nodes.

The default value of this parameter is set to 256KB. Its maximum size is 16MB.

- **TotalSendBufferMemory**

  This parameter is used to determine the total amount of memory to allocate on this node for shared send buffer memory among all configured transporters.

  If this parameter is set, its minimum permitted value is 256KB; 0 indicates that the parameter has not been set. For more detailed information, see Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

- **AutoReconnect**

  This parameter is `false` by default. This forces disconnected API nodes (including MySQL Servers acting as SQL nodes) to use a new connection to the cluster rather than attempting to re-use an existing one, as re-use of connections can cause problems when using dynamically-allocated node IDs. (Bug #45921)

  **Note**

  This parameter can be overridden using the NDB API. For more information, see `Ndb_cluster_connection::set_auto_reconnect()`, and `Ndb_cluster_connection::get_auto_reconnect()`.

- **DefaultOperationRedoProblemAction**

  This parameter (along with `RedoOverCommitLimit` and `RedoOverCommitCounter`) controls the data node’s handling of operations when too much time is taken flushing redo logs to disk. This occurs when a given redo log flush takes longer than `RedoOverCommitLimit` seconds, more than `RedoOverCommitCounter` times, causing any pending transactions to be aborted.

  When this happens, the node can respond in either of two ways, according to the value of `DefaultOperationRedoProblemAction`, listed here:

  - **ABORT**: Any pending operations from aborted transactions are also aborted.
  - **QUEUE**: Pending operations from transactions that were aborted are queued up to be re-tried. This the default. Pending operations are still aborted when the redo log runs out of space—that is, when `P_TAIL_PROBLEM` errors occur.

- **DefaultHashMapSize**

  The size of the table hash maps used by NDB is configurable using this parameter. `DefaultHashMapSize` can take any of three possible values (0, 240, 3840). These values and their effects are described in the following table.

  **Table 5.10 DefaultHashMapSize parameter values**

<table>
<thead>
<tr>
<th>Value</th>
<th>Description / Effect</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>Use the lowest value set, if any, for this parameter among all data nodes and API nodes in the cluster; if it is not set on any data or API node, use the default value.</td>
</tr>
</tbody>
</table>
### Value | Description / Effect
--- | ---
240 | Old default hash map size
3840 | Hash map size used by default in NDB 8.0

The original intended use for this parameter was to facilitate upgrades and downgrades to and from older NDB Cluster versions, in which the hash map size differed, due to the fact that this change was not otherwise backward compatible. This is not an issue when upgrading to or downgrading from NDB Cluster 8.0.

- **Wan**

  Use WAN TCP setting as default.

- **ConnectBackoffMaxTime**

  In an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

  Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `ConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

  Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

- **StartConnectBackoffMaxTime**

  In an NDB Cluster with many unstarted data nodes, the value of this parameter can be raised to circumvent connection attempts to data nodes which have not yet begun to function in the cluster, as well as moderate high traffic to management nodes. As long as the API node is not connected to any new data nodes, the value of the `StartConnectBackoffMaxTime` parameter is applied; otherwise, `ConnectBackoffMaxTime` is used to determine the length of time in milliseconds to wait between connection attempts.

  Time elapsed *during* node connection attempts is not taken into account when calculating elapsed time for this parameter. The timeout is applied with approximately 100 ms resolution, starting with a 100 ms delay; for each subsequent attempt, the length of this period is doubled until it reaches `StartConnectBackoffMaxTime` milliseconds, up to a maximum of 100000 ms (100s).

  Once the API node is connected to a data node and that node reports (in a heartbeat message) that it has connected to other data nodes, connection attempts to those data nodes are no longer affected by this parameter, and are made every 100 ms thereafter until connected. Once a data node has started, it can take up `HeartbeatIntervalDbApi` for the API node to be notified that this has occurred.

**API Node Debugging Parameters.** You can use the `ApiVerbose` configuration parameter to enable debugging output from a given API node. This parameter takes an integer value. 0 is the default, and disables such debugging; 1 enables debugging output to the cluster log; 2 adds DBDICT debugging output as well. (Bug #20638450) See also DUMP 1229.
You can also obtain information from a MySQL server running as an NDB Cluster SQL node using `SHOW STATUS` in the `mysql` client, as shown here:

```sql
mysql> SHOW STATUS LIKE 'ndb%';
+-----------------------------+----------------+
| Variable_name               | Value          |
+-----------------------------+----------------+
| Ndb_cluster_node_id         | 5              |
| Ndb_config_from_host        | 198.51.100.112 |
| Ndb_config_from_port        | 1186           |
| Ndb_number_of_storage_nodes | 4              |
+-----------------------------+----------------+
4 rows in set (0.02 sec)
```

For information about the status variables appearing in the output from this statement, see Section 5.3.9.3, "NDB Cluster Status Variables".

**Note**

To add new SQL or API nodes to the configuration of a running NDB Cluster, it is necessary to perform a rolling restart of all cluster nodes after adding new `[mysqld]` or `[api]` sections to the `config.ini` file (or files, if you are using more than one management server). This must be done before the new SQL or API nodes can connect to the cluster.

It is not necessary to perform any restart of the cluster if new SQL or API nodes can employ previously unused API slots in the cluster configuration to connect to the cluster.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the <code>--initial</code> option</td>
</tr>
</tbody>
</table>

### 5.3.8 Defining the System

The `[system]` section is used for parameters applying to the cluster as a whole. The `Name` system parameter is used with MySQL Enterprise Monitor; `ConfigGenerationNumber` and `PrimaryMGMNode` are not used in production environments. Except when using NDB Cluster with MySQL Enterprise Monitor, it is not necessary to have a `[system]` section in the `config.ini` file.

More information about these parameters can be found in the following list:

- `ConfigGenerationNumber`

  Configuration generation number. This parameter is currently unused.

- `Name`

  Set a name for the cluster. This parameter is required for deployments with MySQL Enterprise Monitor; it is otherwise unused.
You can obtain the value of this parameter by checking the `ndb_system_name` status variable. In NDB API applications, you can also retrieve it using `get_system_name()`.

- **PrimaryMGMNode**

  Node ID of the primary management node. This parameter is currently unused.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the <code>--initial</code> option</td>
</tr>
</tbody>
</table>

### 5.3.9 MySQL Server Options and Variables for NDB Cluster

This section provides information about MySQL server options, server and status variables that are specific to NDB Cluster. For general information on using these, and for other options and variables not specific to NDB Cluster, see The MySQL Server.

For NDB Cluster configuration parameters used in the cluster configuration file (usually named `config.ini`), see Chapter 5, Configuration of NDB Cluster.

#### 5.3.9.1 MySQL Server Options for NDB Cluster

This section provides descriptions of `mysqld` server options relating to NDB Cluster. For information about `mysqld` options not specific to NDB Cluster, and for general information about the use of options with `mysqld`, see Server Command Options.

For information about command-line options used with other NDB Cluster processes (`ndbd`, `ndb_mgmd`, and `ndb_mgm`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”. For information about command-line options used with NDB utility programs (such as `ndb_desc`, `ndb_size.pl`, and `ndb_show_tables`), see Chapter 6, NDB Cluster Programs.

- **--ndbcluster**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndbcluster[=value]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Disabled by</td>
<td><code>skip-ndbcluster</code></td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td><code>ON</code></td>
</tr>
<tr>
<td>Valid Values</td>
<td><code>OFF</code> <code>FORCE</code></td>
</tr>
</tbody>
</table>

The `NDBCLUSTER` storage engine is necessary for using NDB Cluster. If a `mysqld` binary includes support for the `NDBCLUSTER` storage engine, the engine is disabled by default. Use the `--ndbcluster` option to enable it. Use `--skip-ndbcluster` to explicitly disable the engine.
The `--ndbcluster` option is ignored (and the NDB storage engine is not enabled) if `--initialize` is also used. (It is neither necessary nor desirable to use this option together with `--initialize`.)

- **--ndb-allow-copying-alter-table= [ON|OFF]**

  | Command-Line Format | --ndb-allow-copying-alter-table={OFF|ON} |
  |---------------------|----------------------------------------|
  | System Variable     | ndb_allow_copying_alter_table          |
  | Scope               | Global, Session                       |
  | Dynamic             | Yes                                    |
  | SET_VAR Hint Applies| No                                     |
  | Type                | Boolean                                |
  | Default Value       | ON                                     |

  Let `ALTER TABLE` and other DDL statements use copying operations on NDB tables. Set to OFF to keep this from happening; doing so may improve performance of critical applications.

- **--ndb-batch-size=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-batch-size</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_batch_size</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>32768</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>31536000</td>
</tr>
</tbody>
</table>

  This sets the size in bytes that is used for NDB transaction batches.

- **--ndb-cluster-connection-pool=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-cluster-connection-pool</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_cluster_connection_pool</td>
</tr>
<tr>
<td>System Variable</td>
<td>ndb_cluster_connection_pool</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
</tbody>
</table>
By setting this option to a value greater than 1 (the default), a `mysqld` process can use multiple connections to the cluster, effectively mimicking several SQL nodes. Each connection requires its own `[api]` or `[mysqld]` section in the cluster configuration (`config.ini`) file, and counts against the maximum number of API connections supported by the cluster.

Suppose that you have 2 cluster host computers, each running an SQL node whose `mysqld` process was started with `--ndb-cluster-connection-pool=4`; this means that the cluster must have 8 API slots available for these connections (instead of 2). All of these connections are set up when the SQL node connects to the cluster, and are allocated to threads in a round-robin fashion.

This option is useful only when running `mysqld` on host machines having multiple CPUs, multiple cores, or both. For best results, the value should be smaller than the total number of cores available on the host machine. Setting it to a value greater than this is likely to degrade performance severely.

**Important**

Because each SQL node using connection pooling occupies multiple API node slots—each slot having its own node ID in the cluster—you must not use a node ID as part of the cluster connection string when starting any `mysqld` process that employs connection pooling. Setting a node ID in the connection string when using the `--ndb-cluster-connection-pool` option causes node ID allocation errors when the SQL node attempts to connect to the cluster.

```
• `--ndb-cluster-connection-pool-nodeids=list`
```

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-cluster-connection-pool-nodeids</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_cluster_connection_pool_nodeids</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Set</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

Specifies a comma-separated list of node IDs for connections to the cluster used by an SQL node. The number of nodes in this list must be the same as the value set for the `--ndb-cluster-connection-pool` option.

```
• `--ndb-blob-read-batch-bytes=bytes`
```

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-blob-read-batch-bytes</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_blob_read_batch_bytes</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>65536</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
</tbody>
</table>
This option can be used to set the size (in bytes) for batching of BLOB data reads in NDB Cluster applications. When this batch size is exceeded by the amount of BLOB data to be read within the current transaction, any pending BLOB read operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling BLOB read batching.

**Note**

In NDB API applications, you can control BLOB write batching with the `setMaxPendingBlobReadBytes()` and `getMaxPendingBlobReadBytes()` methods.

```
--ndb-blob-write-batch-bytes=bytes
```

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-blob-write-batch-bytes</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_blob_write_batch_bytes</td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>65536</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

This option can be used to set the size (in bytes) for batching of BLOB data writes in NDB Cluster applications. When this batch size is exceeded by the amount of BLOB data to be written within the current transaction, any pending BLOB write operations are immediately executed.

The maximum value for this option is 4294967295; the default is 65536. Setting it to 0 has the effect of disabling BLOB write batching.

**Note**

In NDB API applications, you can control BLOB write batching with the `setMaxPendingBlobWriteBytes()` and `getMaxPendingBlobWriteBytes()` methods.

```
--ndb-connectstring=connection_string
```

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-connectstring</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
</tbody>
</table>

When using the NDBCLUSTER storage engine, this option specifies the management server that distributes cluster configuration data. See Section 5.3.3, “NDB Cluster Connection Strings”, for syntax.

```
--ndb-default-column-format={FIXED|DYNAMIC}
```

| Command-Line Format | --ndb-default-column-format={FIXED|DYNAMIC} |
|---------------------|-------------------------------------------|
| System Variable     | ndb_default_column_format                 |
| Scope               | Global                                    |
| Dynamic             | Yes                                       |
**SET_VAR** Hint Applies | No
---|---
Type | Enumeration
Default Value | FIXED
Valid Values
| FIXED
| DYNAMIC

Sets the default COLUMN_FORMAT and ROW_FORMAT for new tables (see [CREATE TABLE Statement](https://dev.mysql.com/doc/refman/8.0/en/create-table.html)). The default is FIXED.

- **--ndb-deferred-constraints=0|1**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-deferred-constraints</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_deferred_constraints</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1</td>
</tr>
</tbody>
</table>

Controls whether or not constraint checks on unique indexes are deferred until commit time, where such checks are supported. 0 is the default.

This option is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

- **--ndb-schema-dist-timeout=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-schema-dist-timeout=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.17-ndb-8.0.17</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>ndb_schema_dist_timeout</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>120</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1200</td>
</tr>
<tr>
<td>Unit</td>
<td>seconds</td>
</tr>
</tbody>
</table>

Specifies the maximum time in seconds that this *mysqld* waits for a schema operation to complete before marking it as having timed out.

- **--ndb-distribution=[KEYHASH|LINHASH]**

| Command-Line Format | `--ndb-distribution={KEYHASH|LINHASH}` |
|---------------------|----------------------------------------|
| System Variable     | `ndb_distribution`                    |
| Scope               | Global                                |
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Dynamic</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>KEYHASH</td>
</tr>
<tr>
<td>Valid Values</td>
<td>LINHASH, KEYHASH</td>
</tr>
</tbody>
</table>

Controls the default distribution method for NDB tables. Can be set to either of **KEYHASH** (key hashing) or **LINHASH** (linear hashing). **KEYHASH** is the default.

- **--ndb-log-apply-status**

  | Command-Line Format | --ndb-log-apply-status[={OFF|ON}] |
  |---------------------|----------------------------------|
  | System Variable     | ndb_log_apply_status            |
  | Scope               | Global                          |
  | Dynamic             | No                              |
  | **SET_VAR** Hint Applies | No     |
  | Type                | Boolean                         |
  | Default Value       | OFF                             |

Causes a replica *mysqld* to log any updates received from its immediate source to the *mysql.ndb_apply_status* table in its own binary log using its own server ID rather than the server ID of the source. In a circular or chain replication setting, this allows such updates to propagate to the *mysql.ndb_apply_status* tables of any MySQL servers configured as replicas of the current *mysqld*.

In a chain replication setup, using this option allows downstream (replica) clusters to be aware of their positions relative to all of their upstream contributors (sources).

In a circular replication setup, this option causes changes to *ndb_apply_status* tables to complete the entire circuit, eventually propagating back to the originating NDB Cluster. This also allows a cluster acting as a replication source to see when its changes (epochs) have been applied to the other clusters in the circle.

This option has no effect unless the MySQL server is started with the **--ndbcluster** option.

- **--ndb-log-empty-epochs=[ON|OFF]**

  | Command-Line Format | --ndb-log-empty-epochs[={OFF|ON}] |
  |---------------------|----------------------------------|
  | System Variable     | ndb_log_empty_epochs             |
  | Scope               | Global                          |
  | Dynamic             | Yes                             |
  | **SET_VAR** Hint Applies | No     |
  | Type                | Boolean                         |
MySQL Server Options and Variables for NDB Cluster

**Default Value**

| OFF |

Causes epochs during which there were no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, even when `log_slave_updates` is enabled.

By default this option is disabled. Disabling `--ndb-log-empty-epochs` causes epoch transactions with no changes not to be written to the binary log, although a row is still written even for an empty epoch in `ndb_binlog_index`.

Because `--ndb-log-empty-epochs=1` causes the size of the `ndb_binlog_index` table to increase independently of the size of the binary log, users should be prepared to manage the growth of this table, even if they expect the cluster to be idle a large part of the time.

• `--ndb-log-empty-update=[ON|OFF]`

**Command-Line Format**

`--ndb-log-empty-update=[{OFF|ON}]`

**System Variable**

`ndb_log_empty_update`

**Scope**

Global

**Dynamic**

Yes

**SET_VAR Hint Applies**

No

**Type**

Boolean

**Default Value**

OFF

Causes updates that produced no changes to be written to the `ndb_apply_status` and `ndb_binlog_index` tables, when when `log_slave_updates` is enabled.

By default this option is disabled (OFF). Disabling `--ndb-log-empty-update` causes updates with no changes not to be written to the binary log.

• `--ndb-log-exclusive-reads=[0|1]`

**Command-Line Format**

`--ndb-log-exclusive-reads=[{OFF|ON}]`

**System Variable**

`ndb_log_exclusive_reads`

**Scope**

Global, Session

**Dynamic**

Yes

**SET_VAR Hint Applies**

No

**Type**

Boolean

**Default Value**

0

Starting the server with this option causes primary key reads to be logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. You can also enable and disable these locks at runtime by setting the value of the `ndb_log_exclusive_reads` system variable to 1 or 0, respectively. 0 (disable locking) is the default.

For more information, see Read conflict detection and resolution.

• `--ndb-log-fail-terminate`

**Command-Line Format**

`--ndb-log-fail-terminate`

**Introduced**

8.0.21-ndb-8.0.21

**System Variable**

`ndb_log_fail_terminate`

**Scope**

Global
When this option is specified, and complete logging of all found row events is not possible, the `mysqld` process is terminated.

- **--ndb-log-orig**

  Command-Line Format: `--ndb-log-orig[={OFF|ON}]`
  System Variable: `ndb_log_orig`
  Scope: Global
  Dynamic: No
  `SET_VAR` Hint Applies: No
  Type: Boolean
  Default Value: OFF

  Log the originating server ID and epoch in the `ndb_binlog_index` table.

  **Note**
  This makes it possible for a given epoch to have multiple rows in `ndb_binlog_index`, one for each originating epoch.

  For more information, see Section 8.4, “NDB Cluster Replication Schema and Tables”.

- **--ndb-log-transaction-id**

  Command-Line Format: `--ndb-log-transaction-id[={OFF|ON}]`
  System Variable: `ndb_log_transaction_id`
  Scope: Global
  Dynamic: No
  `SET_VAR` Hint Applies: No
  Type: Boolean
  Default Value: OFF

  Causes a replica `mysqld` to write the NDB transaction ID in each row of the binary log. The default value is `FALSE`.

  This option is not supported in mainline MySQL Server 8.0. It is required to enable NDB Cluster Replication conflict detection and resolution using the `NDB$EPOCH_TRANS()` function (see `NDB$EPOCH_TRANS()`). For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

  The deprecated `log_bin_use_v1_row_events` system variable, which defaults to `OFF`, must not be set to `ON` when you use `--ndb-log-transaction-id`.

- **--ndb-log-update-minimal**

  Command-Line Format: `--ndb-log-update-minimal[={OFF|ON}]`
  System Variable: `ndb_log_update_minimal`
  Scope: Global
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Dynamic</th>
<th>Yes</th>
</tr>
</thead>
<tbody>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Log updates in a minimal fashion, by writing only the primary key values in the before image, and only the changed columns in the after image. This may cause compatibility problems if replicating to storage engines other than NDB.

- `--ndb-mgmd-host=host[:port]`

  Command-Line Format | `--ndb-mgmd-host=host_name[:port_num]`
  Status Variable | Ndb_cluster_node_id
  Scope | Global
  Dynamic | No
  Type | String
  Default Value | localhost:1186

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the `--ndb-connectstring` option instead.

- `--ndb-nodeid=#`

  Command-Line Format | `--ndb-nodeid=#`
  Status Variable | Ndb_cluster_node_id
  Scope | Global
  Dynamic | No
  Type | Integer
  Minimum Value | 1
  Maximum Value | 255
  Maximum Value | 63

Set this MySQL server's node ID in an NDB Cluster.

The `--ndb-nodeid` option overrides any node ID set with `--ndb-connectstring`, regardless of the order in which the two options are used.

In addition, if `--ndb-nodeid` is used, then either a matching node ID must be found in a [mysqld] or [api] section of config.ini, or there must be an “open” [mysqld] or [api] section in the file (that is, a section without a NodeId or Id parameter specified). This is also true if the node ID is specified as part of the connection string.

Regardless of how the node ID is determined, its is shown as the value of the global status variable Ndb_cluster_node_id in the output of SHOW STATUS, and as cluster_node_id in the connection row of the output of SHOW ENGINE NDBCLUSTER STATUS.

For more information about node IDs for NDB Cluster SQL nodes, see Section 5.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- `--ndbinfo={ON|OFF|FORCE}`

  Command-Line Format | `--ndbinfo=value` (≥ 8.0.13-ndb-8.0.13)
  Introduced | 8.0.13-ndb-8.0.13
  Type | Enumeration
  Default Value | ON
### MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Valid Values</th>
<th>ON</th>
<th>OFF</th>
<th>FORCE</th>
</tr>
</thead>
</table>

Enables the plugin for the `ndbinfo` information database. By default this is ON whenever `NDBCLUSTER` is enabled.

#### `--ndb-optimization-delay=milliseconds`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-optimization-delay=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_optimization_delay</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>10</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>100000</td>
</tr>
</tbody>
</table>

Set the number of milliseconds to wait between sets of rows by `OPTIMIZE TABLE` statements on NDB tables. The default is 10.

#### `--ndb-transid-mysql-connection-map=state`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-transid-mysql-connection-map=[=state]</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
<tr>
<td>Valid Values</td>
<td>ON, OFF, FORCE</td>
</tr>
</tbody>
</table>

Enables or disables the plugin that handles the `ndb_transid_mysql_connection_map` table in the `INFORMATION_SCHEMA` database. Takes one of the values ON, OFF, or FORCE. ON (the default) enables the plugin. OFF disables the plugin, which makes `ndb_transid_mysql_connection_map` inaccessible. FORCE keeps the MySQL Server from starting if the plugin fails to load and start.

You can see whether the `ndb_transid_mysql_connection_map` table plugin is running by checking the output of `SHOW PLUGINS`.

#### `--ndb-wait-connected=seconds`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-wait-connected=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_wait_connected</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>30</td>
</tr>
</tbody>
</table>
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Option</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--ndb-wait-setup</td>
<td>0</td>
<td>31536000</td>
</tr>
</tbody>
</table>

This option sets the period of time that the MySQL server waits for connections to NDB Cluster management and data nodes to be established before accepting MySQL client connections. The time is specified in seconds. The default value is **30**.

- **--ndb-wait-setup=seconds**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-wait-setup=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_wait_setup</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>30</td>
</tr>
<tr>
<td>Default Value</td>
<td>30</td>
</tr>
<tr>
<td>Default Value</td>
<td>15</td>
</tr>
<tr>
<td>Default Value</td>
<td>15</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>31536000</td>
</tr>
</tbody>
</table>

This variable shows the period of time that the MySQL server waits for the **NDB** storage engine to complete setup before timing out and treating **NDB** as unavailable. The time is specified in seconds. The default value is **30**.

- **--skip-ndbcluster**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--skip-ndbcluster</th>
</tr>
</thead>
</table>

Disable the **NDBCLUSTER** storage engine. This is the default for binaries that were built with **NDBCLUSTER** storage engine support; the server allocates memory and other resources for this storage engine only if the **--ndbcluster** option is given explicitly. See Section 5.1, “Quick Test Setup of NDB Cluster”, for an example.

### 5.3.9.2 NDB Cluster System Variables

This section provides detailed information about MySQL server system variables that are specific to NDB Cluster and the **NDB** storage engine. For system variables not specific to NDB Cluster, see **Server System Variables**. For general information on using system variables, see **Using System Variables**.

- **ndb_autoincrement_prefetch_sz**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-autoincrement-prefetch-sz=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_autoincrement_prefetch_sz</td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value (≥ 8.0.19-ndb-8.0.19)</td>
<td>512</td>
</tr>
</tbody>
</table>
Determines the probability of gaps in an autoincremented column. Set it to 1 to minimize this. Setting it to a high value for optimization makes inserts faster, but decreases the likelihood of consecutive autoincrement numbers being used in a batch of inserts.

This variable affects only the number of AUTO_INCREMENT IDs that are fetched between statements; within a given statement, at least 32 IDs are obtained at a time.

**Important**

This variable does not affect inserts performed using `INSERT ... SELECT`.

- **ndb_cache_check_time**

  Command-Line Format: `--ndb-cache-check-time=#`

  Deprecated: Yes

  System Variable: `ndb_cache_check_time`

  Scope: Global

  Dynamic: Yes

  SET_VAR: No

  Type: Integer

  Default Value: 0

  The number of milliseconds that elapse between checks of NDB Cluster SQL nodes by the MySQL query cache. Setting this to 0 (the default and minimum value) means that the query cache checks for validation on every query.

  The recommended maximum value for this variable is 1000, which means that the check is performed once per second. A larger value means that the check is performed and possibly invalidated due to updates on different SQL nodes less often. It is generally not desirable to set this to a value greater than 2000.

**Note**

The query cache `ndb_cache_check_time` are deprecated in MySQL 5.7; the query cache was removed in MySQL 8.0.

- **ndb_clear_apply_status**

  Command-Line Format: `--ndb-clear-apply-status=[{OFF|ON}]`

  System Variable: `ndb_clear_apply_status`

  Scope: Global

  Dynamic: Yes

  SET_VAR: No

  Type: Boolean

  Default Value: ON

  By the default, executing `RESET SLAVE` causes an NDB Cluster replica to purge all rows from its `ndb_apply_status` table. You can disable this by setting `ndb_clear_apply_status=OFF`. 
• **ndb_conflict_role**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-conflict-role=value</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.23-ndb-8.0.23</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>ndb_conflict_role</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>NONE</td>
</tr>
<tr>
<td>Valid Values</td>
<td>NONE, PRIMARY, SECONDARY, PASS</td>
</tr>
</tbody>
</table>

Determines the role of this SQL node (and NDB Cluster) in a circular (“active-active”) replication setup. `ndb_slave_conflict_role` can take any one of the values PRIMARY, SECONDARY, PASS, or NULL (the default). The replica SQL thread must be stopped before you can change `ndb_slave_conflict_role`. In addition, it is not possible to change directly between PASS and either of PRIMARY or SECONDARY directly; in such cases, you must ensure that the SQL thread is stopped, then execute `SET @@GLOBAL.ndb_slave_conflict_role = 'NONE'` first.

This variable replaces `ndb_slave_conflict_role`, which is deprecated as of NDB 8.0.23.

For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **ndb_data_node_neighbour**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-data-node-neighbour=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_data_node_neighbour</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>255</td>
</tr>
</tbody>
</table>

Sets the ID of a “nearest” data node—that is, a preferred nonlocal data node is chosen to execute the transaction, rather than one running on the same host as the SQL or API node. This used to ensure that when a fully replicated table is accessed, we access it on this data node, to ensure that
the local copy of the table is always used whenever possible. This can also be used for providing hints for transactions.

This can improve data access times in the case of a node that is physically closer than and thus has higher network throughput than others on the same host.

See Setting NDB_TABLE Options, for further information.

### Note

An equivalent method `set_data_node_neighbour()` is provided for use in NDB API applications.

• **ndb_dbg_check_shares**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-dbg-check-shares=##</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.13-ndb-8.0.13</td>
</tr>
<tr>
<td>System Variable</td>
<td>ndb_dbg_check_shares</td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1</td>
</tr>
</tbody>
</table>

When set to 1, check that no shares are lingering. Available in debug builds only.

Added in NDB 8.0.13.

• **ndb_default_column_format**

| Command-Line Format | --ndb-default-column-format={FIXED|DYNAMIC} |
|---------------------|------------------------------------------|
| System Variable     | ndb_default_column_format                |
| Scope               | Global                                   |
| Dynamic             | Yes                                      |
| SET_VAR Hint Applies| No                                       |
| Type                | Enumeration                              |
| Default Value       | FIXED                                    |
| Valid Values        | FIXED, DYNAMIC                           |

Sets the default `COLUMN_FORMAT` and `ROW_FORMAT` for new tables (see CREATE TABLE Statement). The default is `FIXED`.

• **ndb_deferred_constraints**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-deferred-constraints=##</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_deferred_constraints</td>
</tr>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>
## MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>SET_VAR</th>
<th>Hint Applies</th>
<th>Type</th>
<th>Default Value</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>Integer</td>
<td>0</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Controls whether or not constraint checks are deferred, where these are supported. 0 is the default.

This variable is not normally needed for operation of NDB Cluster or NDB Cluster Replication, and is intended primarily for use in testing.

### ndb_distribution

| Command-Line Format | --ndb-distribution={KEYHASH|LINHASH} |
|---------------------|--------------------------------------|
| System Variable     | ndb_distribution                     |
| Scope               | Global                               |
| Dynamic             | Yes                                  |

Controls the default distribution method for NDB tables. Can be set to either of KEYHASH (key hashing) or LINHASH (linear hashing). KEYHASH is the default.

### ndb_eventbuffer_free_percent

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-eventbuffer-free-percent=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_eventbuffer_free_percent</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>

Sets the percentage of the maximum memory allocated to the event buffer (ndb_eventbuffer_max_alloc) that should be available in event buffer after reaching the maximum, before starting to buffer again.

### ndb_eventbuffer_max_alloc

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-eventbuffer-max-alloc=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_eventbuffer_max_alloc</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>SET_VAR</th>
<th>Hint Applies</th>
<th>Type</th>
<th>Default Value</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>No</td>
<td>Integer</td>
<td>20</td>
<td>1</td>
<td>99</td>
</tr>
</tbody>
</table>

Sets the percentage of the maximum memory allocated to the event buffer (ndb_eventbuffer_max_alloc) that should be available in event buffer after reaching the maximum, before starting to buffer again.
### MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Type</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Sets the maximum amount memory (in bytes) that can be allocated for buffering events by the NDB API. 0 means that no limit is imposed, and is the default.

- **ndb_extra_logging**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>ndb_extra_logging=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndb_extra_logging</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
</tbody>
</table>

This variable enables recording in the MySQL error log of information specific to the NDB storage engine.

When this variable is set to 0, the only information specific to NDB that is written to the MySQL error log relates to transaction handling. If it set to a value greater than 0 but less than 10, NDB table schema and connection events are also logged, as well as whether or not conflict resolution is in use, and other NDB errors and information. If the value is set to 10 or more, information about NDB internals, such as the progress of data distribution among cluster nodes, is also written to the MySQL error log. The default is 1.

- **ndb_force_send**

  | Command-Line Format | --ndb-force-send[={OFF|ON}] |
  |---------------------|-----------------------------|
  | System Variable     | ndb_force_send              |
  | Scope               | Global, Session             |
  | Dynamic             | Yes                         |
  | SET_VAR Hint Applies| No                          |
  | Type                | Boolean                     |
  | Default Value       | ON                          |

Forces sending of buffers to NDB immediately, without waiting for other threads. Defaults to ON.

- **ndb_fully_replicated**

  | Command-Line Format | --ndb-fully-replicated[={OFF|ON}] |
  |---------------------|-----------------------------------|
  | System Variable     | ndb_fully_replicated             |
  | Scope               | Global, Session                  |
  | Dynamic             | Yes                               |
  | SET_VAR Hint Applies| No                               |
  | Type                | Boolean                          |
  | Default Value       | ON                               |
Determines whether new NDB tables are fully replicated. This setting can be overridden for an individual table using `COMMENT="NDB_TABLE=FULLY_REPLICATED=..."` in a `CREATE TABLE` or `ALTER TABLE` statement; see Setting NDB_TABLE Options, for syntax and other information.

- **ndb_index_stat_enable**

  **Command-Line Format**
  
  `--ndb-index-stat-enable[={OFF|ON}]`

  **System Variable**
  
  `ndb_index_stat_enable`

  **Scope**
  
  Global, Session

  **Dynamic**
  
  Yes

  **SET_VAR Hint Applies**
  
  No

  **Type**
  
  Boolean

  **Default Value**
  
  ON

  Use NDB index statistics in query optimization. The default is ON.

- **ndb_index_stat_option**

  **Command-Line Format**
  
  `--ndb-index-stat-option=value`

  **System Variable**
  
  `ndb_index_stat_option`

  **Scope**
  
  Global, Session

  **Dynamic**
  
  Yes

  **SET_VAR Hint Applies**
  
  No

  **Type**
  
  String

  **Default Value**
  
  `loop_checkon=1000ms,loop_idle=1000ms,loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=32,check_delay=1m,delete_batch=8,clean_delay=0,error_batch=4,error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90`

  This variable is used for providing tuning options for NDB index statistics generation. The list consists of comma-separated name-value pairs of option names and values, and this list must not contain any space characters.

  Options not used when setting `ndb_index_stat_option` are not changed from their default values. For example, you can set `ndb_index_stat_option = 'loop_idle=1000ms,cache_limit=32M'`.

  Time values can be optionally suffixed with `h` (hours), `m` (minutes), or `s` (seconds). Millisecond values can optionally be specified using `ms`; millisecond values cannot be specified using `h`, `m`, or `s`). Integer values can be suffixed with `K`, `M`, or `G`.

  The names of the options that can be set using this variable are shown in the table that follows. The table also provides brief descriptions of the options, their default values, and (where applicable) their minimum and maximum values.

  **Table 5.13 ndb_index_stat_option options and values**

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default/Units</th>
<th>Minimum/Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>loop_enable</code></td>
<td>1000 ms</td>
<td></td>
<td>0/4G</td>
</tr>
<tr>
<td><code>loop_idle</code></td>
<td>Time to sleep when idle</td>
<td>1000 ms</td>
<td>0/4G</td>
</tr>
</tbody>
</table>
### MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Name</th>
<th>Description</th>
<th>Default/Units</th>
<th>Minimum/Maximum</th>
</tr>
</thead>
<tbody>
<tr>
<td>loop_busy</td>
<td>Time to sleep when more work is waiting</td>
<td>100 ms</td>
<td>0/4G</td>
</tr>
<tr>
<td>update_batch</td>
<td></td>
<td>1</td>
<td>0/4G</td>
</tr>
<tr>
<td>read_batch</td>
<td></td>
<td>4</td>
<td>1/4G</td>
</tr>
<tr>
<td>idle_batch</td>
<td></td>
<td>32</td>
<td>1/4G</td>
</tr>
<tr>
<td>check_batch</td>
<td></td>
<td>8</td>
<td>1/4G</td>
</tr>
<tr>
<td>check_delay</td>
<td>How often to check for new statistics</td>
<td>10 m</td>
<td>1/4G</td>
</tr>
<tr>
<td>delete_batch</td>
<td></td>
<td>8</td>
<td>0/4G</td>
</tr>
<tr>
<td>clean_delay</td>
<td></td>
<td>1 m</td>
<td>0/4G</td>
</tr>
<tr>
<td>error_batch</td>
<td></td>
<td>4</td>
<td>1/4G</td>
</tr>
<tr>
<td>error_delay</td>
<td></td>
<td>1 m</td>
<td>1/4G</td>
</tr>
<tr>
<td>evict_batch</td>
<td></td>
<td>8</td>
<td>1/4G</td>
</tr>
<tr>
<td>evict_delay</td>
<td>Clean LRU cache, from read time</td>
<td>1 m</td>
<td>0/4G</td>
</tr>
<tr>
<td>cache_limit</td>
<td>Maximum amount of memory in bytes used for cached index statistics by this mysqld; clean up the cache when this is exceeded.</td>
<td>32 M</td>
<td>0/4G</td>
</tr>
<tr>
<td>cache_lowpct</td>
<td>90</td>
<td>0/100</td>
<td></td>
</tr>
<tr>
<td>zero_total</td>
<td>Setting this to 1 resets all accumulating counters in ndb_index_stat_status to 0. This option value is also reset to 0 when this is done.</td>
<td>0</td>
<td>0/1</td>
</tr>
</tbody>
</table>

- **ndb_join_pushdown**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_join_pushdown</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>ON</td>
</tr>
</tbody>
</table>

This variable controls whether joins on NDB tables are pushed down to the NDB kernel (data nodes). Previously, a join was handled using multiple accesses of NDB by the SQL node; however, when **ndb_join_pushdown** is enabled, a pushable join is sent in its entirety to the data nodes, where it can be distributed among the data nodes and executed in parallel on multiple copies of the data, with
a single, merged result being returned to mysqld. This can reduce greatly the number of round trips between an SQL node and the data nodes required to handle such a join.

By default, ndb_join_pushdown is enabled.

**Conditions for NDB pushdown joins.** In order for a join to be pushable, it must meet the following conditions:

1. Only columns can be compared, and all columns to be joined must use exactly the same data type. This means that (for example) a join on an INT column and a BIGINT column also cannot be pushed down.

   Previously, expressions such as $t1.a = t2.a + constant$ could not be pushed down. This restriction is lifted in NDB 8.0.18 and later. The result of any operations on any column to be compared must yield the same type as the column itself.

   Also beginning with NDB 8.0.18, expressions comparing columns from the same table can be pushed down. The columns (or the result of any operations on those columns) must be of exactly the same type, including the same signedness, length, character set and collation, precision, and scale, where these are applicable.

2. Queries referencing BLOB or TEXT columns are not supported.

3. Explicit locking is not supported; however, the NDB storage engine’s characteristic implicit row-based locking is enforced.

   This means that a join using FOR UPDATE cannot be pushed down.

4. In order for a join to be pushed down, child tables in the join must be accessed using one of the ref, eq_ref, or const access methods, or some combination of these methods.

   Outer joined child tables can only be pushed using eq_ref.

   If the root of the pushed join is an eq_ref or const, only child tables joined by eq_ref can be appended. (A table joined by ref is likely to become the root of another pushed join.)

   If the query optimizer decides on **Using join cache** for a candidate child table, that table cannot be pushed as a child. However, it may be the root of another set of pushed tables.

5. Joins referencing tables explicitly partitioned by [LINEAR] HASH, LIST, or RANGE currently cannot be pushed down.

You can see whether a given join can be pushed down by checking it with **EXPLAIN**: when the join can be pushed down, you can see references to the pushed join in the Extra column of the output, as shown in this example:

```sql
mysql> EXPLAIN
    -> SELECT e.first_name, e.last_name, t.title, d.dept_name
    -> FROM employees e
    -> JOIN dept_emp de ON e.emp_no=de.emp_no
    -> JOIN departments d ON d.dept_no=de.dept_no
    -> JOIN titles t ON e.emp_no=t.emp_no\G
```

```
*************************** 1. row ***************************
  id: 1
  select_type: SIMPLE
table: d
  type: ALL
  possible_keys: PRIMARY
  key: NULL
  key_len: NULL
  ref: NULL
  rows: 9
  Extra: Parent of 4 pushed join@1

*************************** 2. row ***************************
```

201
MySQL Server Options and Variables for NDB Cluster

Note

If inner joined child tables are joined by `ref`, and the result is ordered or grouped by a sorted index, this index cannot provide sorted rows, which forces writing to a sorted tempfile.

Two additional sources of information about pushed join performance are available:

1. The status variables `Ndb_pushed_queries_defined`, `Ndb_pushed_queries_dropped`, `Ndb_pushed_queries_executed`, and `Ndb_pushed_reads`.

2. The counters in the `ndbinfo.counters` table that belong to the `DBSPJ` kernel block.

   • `ndb_log_apply_status`

   | Command-Line Format | `--ndb-log-apply-status[={OFF|ON}]` |
   |---------------------|-------------------------------------|
   | System Variable     | `ndb_log_apply_status`              |
   | Scope               | Global                              |
   | Dynamic             | No                                  |
   | `SET_VAR` Hint Applies | No                                |
   | Type                | Boolean                             |
   | Default Value       | OFF                                 |

   A read-only variable which shows whether the server was started with the `--ndb-log-apply-status` option.

   • `ndb_log_bin`

   | Command-Line Format | `--ndb-log-bin[={OFF|ON}]` |

202
### MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_log_bin</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><strong>SET_VAR</strong> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
<tr>
<td>(≥ 8.0.16-ndb-8.0.16)</td>
<td>OFF</td>
</tr>
<tr>
<td>(≤ 8.0.15-ndb-8.0.15)</td>
<td>ON</td>
</tr>
</tbody>
</table>

Causes updates to NDB tables to be written to the binary log. The setting for this variable has no effect if binary logging is not already enabled on the server using log_bin. In NDB 8.0.16 and later, ndb_log_bin defaults to 0 (FALSE).

- **ndb_log_binlog_index**

  | Command-Line Format | --ndb-log-binlog-index={OFF|ON} |
  |--------------------|-------------------------------|
  | System Variable   | ndb_log_binlog_index          |
  | Scope              | Global                        |
  | Dynamic            | Yes                           |
  | **SET_VAR** Hint Applies | No                  |
  | Type               | Boolean                       |
  | Default Value      | ON                            |

Causes a mapping of epochs to positions in the binary log to be inserted into the ndb_binlog_index table. Setting this variable has no effect if binary logging is not already enabled for the server using log_bin. (In addition, ndb_log_bin must not be disabled.) ndb_log_binlog_index defaults to 1 (ON); normally, there is never any need to change this value in a production environment.

- **ndb_log_empty_epochs**

  | Command-Line Format | --ndb-log-empty-epochs={OFF|ON} |
  |--------------------|-------------------------------|
  | System Variable   | ndb_log_empty_epochs          |
  | Scope              | Global                        |
  | Dynamic            | Yes                           |
  | **SET_VAR** Hint Applies | No                  |
  | Type               | Boolean                       |
  | Default Value      | OFF                           |

When this variable is set to 0, epoch transactions with no changes are not written to the binary log, although a row is still written even for an empty epoch in ndb_binlog_index.

- **ndb_log_empty_update**

  | Command-Line Format | --ndb-log-empty-update={OFF|ON} |
  |--------------------|-------------------------------|
  | System Variable   | ndb_log_empty_update          |
  | Scope              | Global                        |
  | Dynamic            | Yes                           |
  | **SET_VAR** Hint Applies | No                  |
### MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

When this variable is set to **ON** (1), update transactions with no changes are written to the binary log, even when `log_slave_updates` is enabled.

- **ndb_log_exclusive_reads**

  | Command-Line Format          | `--ndb-log-exclusive-reads[={OFF|ON}]` |
  |------------------------------|----------------------------------------|
  | System Variable              | `ndb_log_exclusive_reads`              |
  | Scope                        | Global, Session                       |
  | Dynamic                      | Yes                                    |
  | `SET_VAR` Hint Applies       | No                                     |
  | Type                         | Boolean                               |
  | Default Value                | 0                                      |

This variable determines whether primary key reads are logged with exclusive locks, which allows for NDB Cluster Replication conflict detection and resolution based on read conflicts. To enable these locks, set the value of `ndb_log_exclusive_reads` to 1. 0, which disables such locking, is the default.

For more information, see [Read conflict detection and resolution](#).

- **ndb_log_orig**

  | Command-Line Format          | `--ndb-log-orig[={OFF|ON}]` |
  |------------------------------|-----------------------------|
  | System Variable              | `ndb_log_orig`              |
  | Scope                        | Global                      |
  | Dynamic                      | No                          |
  | `SET_VAR` Hint Applies       | No                          |
  | Type                         | Boolean                     |
  | Default Value                | OFF                         |

Shows whether the originating server ID and epoch are logged in the `ndb_binlog_index` table. Set using the `--ndb-log-orig` server option.

- **ndb_log_transaction_id**

<table>
<thead>
<tr>
<th>System Variable</th>
<th><code>ndb_log_transaction_id</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

This read-only, Boolean system variable shows whether a replica `mysqld` writes NDB transaction IDs in the binary log (required to use “active-active” NDB Cluster Replication with NDB...
$EPOCH_TRANS() conflict detection). To change the setting, use the `--ndb-log-transaction-id` option.

`ndb_log_transaction_id` is not supported in mainline MySQL Server 8.0.

For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- **ndb_metadata_check**

| Command-Line Format | `--ndb-metadata-check[={OFF|ON}]` |
|---------------------|-----------------------------------|
| Introduced          | 8.0.16-ndb-8.0.16                 |
| System Variable     | `ndb_metadata_check`              |
| Scope               | Global                            |
| Dynamic             | Yes                               |
| `SET_VAR` Hint Applies | No                             |
| Type                | Boolean                           |
| Default Value       | ON                                |

Beginning with NDB 8.0.16, NDB uses a background thread to check for metadata changes each `ndb_metadata_check_interval` seconds as compared with the MySQL data dictionary. This metadata change detection thread can be disabled by setting `ndb_metadata_check` to `OFF`. The thread is enabled by default.

- **ndb_metadata_check_interval**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-metadata-check-interval=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.16-ndb-8.0.16</td>
</tr>
<tr>
<td>System Variable</td>
<td><code>ndb_metadata_check_interval</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>60</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>31536000</td>
</tr>
<tr>
<td>Unit</td>
<td>seconds</td>
</tr>
</tbody>
</table>

In NDB 8.0.16 and later, NDB runs a metadata change detection thread in the background to determine when the NDB dictionary has changed with respect to the MySQL data dictionary. By default, the interval between such checks is 60 seconds; this can be adjusted by setting the value of `ndb_metadata_check_interval`. To enable or disable the thread, use `ndb_metadata_check`.

- **ndb_metadata_sync**

| Introduced          | 8.0.19-ndb-8.0.19                 |
| System Variable     | `ndb_metadata_sync`              |
| Scope               | Global                            |
| Dynamic             | Yes                               |
| `SET_VAR` Hint Applies | No                             |
| Type                | Boolean                           |
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Default Value</th>
<th>false</th>
</tr>
</thead>
</table>

Setting this variable causes the change monitor thread to override any values set for `ndb_metadata_check` or `ndb_metadata_check_interval`, and to enter a period of continuous change detection. When the thread ascertains that there are no more changes to be detected, it stalls until the binary logging thread has finished synchronization of all detected objects. `ndb_metadata_sync` is then set to `false`, and the change monitor thread reverts to the behavior determined by the settings for `ndb_metadata_check` and `ndb_metadata_check_interval`.

In NDB 8.0.22 and later, setting this variable to `true` causes the list of excluded objects to be cleared, and setting it to `false` clears the list of objects to be retried.

- `ndb_optimized_node_selection`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-optimized-node-selection=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_optimized_node_selection</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>3</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>3</td>
</tr>
</tbody>
</table>

There are two forms of optimized node selection, described here:

1. The SQL node uses *proximity* to determine the transaction coordinator; that is, the “closest” data node to the SQL node is chosen as the transaction coordinator. For this purpose, a data node having a shared memory connection with the SQL node is considered to be “closest” to the SQL node; the next closest (in order of decreasing proximity) are: TCP connection to `localhost`, followed by TCP connection from a host other than `localhost`.

2. The SQL thread uses *distribution awareness* to select the data node. That is, the data node housing the cluster partition accessed by the first statement of a given transaction is used as the transaction coordinator for the entire transaction. (This is effective only if the first statement of the transaction accesses no more than one cluster partition.)

This option takes one of the integer values 0, 1, 2, or 3. 3 is the default. These values affect node selection as follows:

- **0**: Node selection is not optimized. Each data node is employed as the transaction coordinator 8 times before the SQL thread proceeds to the next data node.
- **1**: Proximity to the SQL node is used to determine the transaction coordinator.
- **2**: Distribution awareness is used to select the transaction coordinator. However, if the first statement of the transaction accesses more than one cluster partition, the SQL node reverts to the round-robin behavior seen when this option is set to 0.
- **3**: If distribution awareness can be employed to determine the transaction coordinator, then it is used; otherwise proximity is used to select the transaction coordinator. (This is the default behavior.)

Proximity is determined as follows:

1. Start with the value set for the `Group` parameter (default 55).
2. For an API node sharing the same host with other API nodes, decrement the value by 1. Assuming the default value for `Group`, the effective value for data nodes on same host as the API node is 54, and for remote data nodes 55.

3. Setting `ndb_data_node_neighbour` further decreases the effective `Group` value by 50, causing this node to be regarded as the nearest node. This is needed only when all data nodes are on hosts other than that hosts the API node and it is desirable to dedicate one of them to the API node. In normal cases, the default adjustment described previously is sufficient.

Frequent changes in `ndb_data_node_neighbour` are not advisable, since this changes the state of the cluster connection and thus may disrupt the selection algorithm for new transactions from each thread until it stabilizes.

- **ndb_read_backup**
  
  | Command-Line Format | `--ndb-read-backup=[{OFF|ON}]` |
  |---------------------|---------------------------------|
  | System Variable    | `ndb_read_backup`               |
  | Scope               | Global                          |
  | Dynamic             | Yes                             |
  | `SET_VAR` Hint Applies | No                              |
  | Type                | Boolean                         |
  | Default Value (≥8.0.19-ndb-8.0.19) | `ON`                          |
  | Default Value (≤8.0.18-ndb-8.0.18) | `OFF`                        |

Enable read from any fragment replica for any NDB table subsequently created; doing so greatly improves the table read performance at a relatively small cost to writes.

To enable or disable read from any fragment replica for an individual table, you can set the `NDB_TABLE` option `READ_BACKUP` for the table accordingly, in a `CREATE TABLE` or `ALTER TABLE` statement; see Setting `NDB_TABLE` Options, for more information.

- **ndb_recv_thread_activation_threshold**
  
<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-recv-thread-activation-threshold=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_recv_thread_activation_threshold</code></td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td><code>SET_VAR</code> Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>8</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0 (MIN_ACTIVATION_THRESHOLD)</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>16 (MAX_ACTIVATION_THRESHOLD)</td>
</tr>
</tbody>
</table>

When this number of concurrently active threads is reached, the receive thread takes over polling of the cluster connection.

This variable is global in scope. It can also be set at startup.

- **ndb_recv_thread_cpu_mask**
  
<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--ndb-recv-thread-cpu-mask=mask</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td><code>ndb_recv_thread_cpu_mask</code></td>
</tr>
</tbody>
</table>
CPU mask for locking receiver threads to specific CPUs. This is specified as a hexadecimal bitmask. For example, 0x33 means that one CPU is used per receiver thread. An empty string is the default; setting `ndb_recv_thread_cpu_mask` to this value removes any receiver thread locks previously set.

This variable is global in scope. It can also be set at startup.

- `ndb_report_thresh_binlog_epoch_slip`

  Command-Line Format: `--ndb-report-thresh-binlog-epoch-slip=#`
  
  System Variable: `ndb_report_thresh_binlog_epoch_slip`
  
  Scope: Global
  
  Dynamic: Yes
  
  SET_VAR: No
  
  Type: Integer
  
  Default Value: 10
  
  Minimum Value: 0
  
  Maximum Value: 256

  This represents the threshold for the number of epochs completely buffered in the event buffer, but not yet consumed by the binlog injector thread. When this degree of slippage (lag) is exceeded, an event buffer status message is reported, with `BUFFERED_EPOCHS_OVER_THRESHOLD` supplied as the reason (see Section 7.2.3, “Event Buffer Reporting in the Cluster Log”). Slip is increased when an epoch is received from data nodes and buffered completely in the event buffer; it is decreased when an epoch is consumed by the binlog injector thread, it is reduced. Empty epochs are buffered and queued, and so included in this calculation only when this is enabled using the `Ndb::setEventBufferQueueEmptyEpoch()` method from the NDB API.

- `ndb_report_thresh_binlog_mem_usage`

  Command-Line Format: `--ndb-report-thresh-binlog-mem-usage=#`
  
  System Variable: `ndb_report_thresh_binlog_mem_usage`
  
  Scope: Global
  
  Dynamic: Yes
  
  SET_VAR: No
  
  Type: Integer
  
  Default Value: 10
  
  Minimum Value: 0
  
  Maximum Value: 10

  This is a threshold on the percentage of free memory remaining before reporting binary log status. For example, a value of 10 (the default) means that if the amount of available memory for receiving binary log data from the data nodes falls below 10%, a status message is sent to the cluster log.
MySQL Server Options and Variables for NDB Cluster

- **ndb_row_checksum**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_row_checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1</td>
</tr>
</tbody>
</table>

Traditionally, NDB has created tables with row checksums, which checks for hardware issues at the expense of performance. Setting `ndb_row_checksum` to 0 means that row checksums are not used for new or altered tables, which has a significant impact on performance for all types of queries. This variable is set to 1 by default, to provide backward-compatible behavior.

- **ndb_schema_dist_lock_wait_timeout**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-schema-dist-lock-wait-timeout=value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.18-ndb-8.0.18</td>
</tr>
<tr>
<td>System Variable</td>
<td>ndb_schema_dist_lock_wait_timeout</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>30</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1200</td>
</tr>
<tr>
<td>Unit</td>
<td>seconds</td>
</tr>
</tbody>
</table>

Number of seconds to wait during schema distribution for the metadata lock taken on each SQL node in order to change its local data dictionary to reflect the DDL statement change. After this time has elapsed, a warning is returned to the effect that a given SQL node's data dictionary was not updated with the change. This avoids having the binary logging thread wait an excessive length of time while handling schema operations.

- **ndb_schema_dist_timeout**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-schema-dist-timeout=value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.16-ndb-8.0.16</td>
</tr>
<tr>
<td>System Variable</td>
<td>ndb_schema_dist_timeout</td>
</tr>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>120</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>5</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1200</td>
</tr>
</tbody>
</table>
Unit | seconds
---|---

Number of seconds to wait before detecting a timeout during schema distribution. This can indicate that other SQL nodes are experiencing excessive activity, or that they are somehow being prevented from acquiring necessary resources at this time.

- **ndb_schema_dist_upgrade_allowed**

  - **Command-Line Format**: `--ndb-schema-dist-upgrade-allowed=value`
  - **Introduced**: 8.0.17-ndb-8.0.17
  - **System Variable**: `ndb_schema_dist_upgrade_allowed`
  - **Scope**: Global
  - **Dynamic**: No
  - **SET_VAR Hint Applies**: No
  - **Type**: Boolean
  - **Default Value**: `true`

  Allow upgrading of the schema distribution table when connecting to NDB. When true (the default), this change is deferred until all SQL nodes have been upgraded to the same version of the NDB Cluster software.

  **Note**

  The performance of the schema distribution may be somewhat degraded until the upgrade has been performed.

- **ndb_show_foreign_key_mock_tables**

  - **Command-Line Format**: `--ndb-show-foreign-key-mock-tables[={OFF|ON}]`
  - **System Variable**: `ndb_show_foreign_key_mock_tables`
  - **Scope**: Global
  - **Dynamic**: Yes
  - **SET_VAR Hint Applies**: No
  - **Type**: Boolean
  - **Default Value**: `OFF`

  Show the mock tables used by NDB to support foreign_key_checks=0. When this is enabled, extra warnings are shown when creating and dropping the tables. The real (internal) name of the table can be seen in the output of `SHOW CREATE TABLE`.

- **ndb_slave_conflict_role**

  - **Command-Line Format**: `--ndb-slave-conflict-role=value`
  - **Deprecated**: 8.0.23-ndb-8.0.23
  - **System Variable**: `ndb_slave_conflict_role`
  - **Scope**: Global
  - **Dynamic**: Yes
  - **SET_VAR Hint Applies**: No
  - **Type**: Enumeration
  - **Default Value**: `NONE`
  - **Valid Values**: `NONE`
**MySQL Server Options and Variables for NDB Cluster**

<table>
<thead>
<tr>
<th>PRIMARY</th>
<th>SECONDARY</th>
<th>PASS</th>
</tr>
</thead>
</table>

Deprecated in NDB 8.0.23, and subject to removal in a future release. Use `ndb_conflict_role` instead.

- **ndb_table_no_logging**

  **System Variable**: `ndb_table_no_logging`
  **Scope**: Session
  **Dynamic**: Yes
  **SET_VAR Hint Applies**: No
  **Type**: Boolean
  **Default Value**: OFF

  When this variable is set to ON or 1, it causes NDB tables not to be checkpointed to disk. More specifically, this setting applies to tables which are created or altered using ENGINE NDB when `ndb_table_no_logging` is enabled, and continues to apply for the lifetime of the table, even if `ndb_table_no_logging` is later changed. Suppose that A, B, C, and D are tables that we create (and perhaps also alter), and that we also change the setting for `ndb_table_no_logging` as shown here:

  ```sql
  SET @@ndb_table_no_logging = 1;
  CREATE TABLE A ... ENGINE NDB;
  CREATE TABLE B ... ENGINE MYISAM;
  CREATE TABLE C ... ENGINE MYISAM;
  ALTER TABLE B ENGINE NDB;
  SET @@ndb_table_no_logging = 0;
  CREATE TABLE D ... ENGINE NDB;
  ALTER TABLE C ENGINE NDB;
  SET @@ndb_table_no_logging = 1;
  ```

  After the previous sequence of events, tables A and B are not checkpointed: A was created with ENGINE NDB and B was altered to use NDB, both while `ndb_table_no_logging` was enabled. However, tables C and D are logged: C was altered to use NDB and D was created using ENGINE NDB, both while `ndb_table_no_logging` was disabled. Setting `ndb_table_no_logging` back to 1 or ON does not cause table C or D to be checkpointed.

  **Note**
  `ndb_table_no_logging` has no effect on the creation of NDB table schema files; to suppress these, use `ndb_table_temporary` instead.

- **ndb_table_temporary**

  **System Variable**: `ndb_table_temporary`
  **Scope**: Session
  **Dynamic**: Yes
  **SET_VAR Hint Applies**: No
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

When set to **ON** or **1**, this variable causes **NDB** tables not to be written to disk: This means that no table schema files are created, and that the tables are not logged.

**Note**

Setting this variable currently has no effect. This is a known issue; see Bug #34036.

- **ndb_use_copying_alter_table**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_use_copying_alter_table</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>

Forces **NDB** to use copying of tables in the event of problems with online **ALTER TABLE** operations. The default value is **OFF**.

- **ndb_use_exact_count**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_use_exact_count</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global, Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

Forces **NDB** to use a count of records during **SELECT COUNT(*)** query planning to speed up this type of query. The default value is **OFF**, which allows for faster queries overall.

- **ndb_use_transactions**

  | Command-Line Format | --ndb-use-transactions[={OFF|ON}] |
  |---------------------|----------------------------------|
  | System Variable     | ndb_use_transactions             |
  | Scope               | Global, Session                  |
  | Dynamic             | Yes                              |
  | SET_VAR Hint Applies | No                         |
  | Type                | Boolean                          |
  | Default Value       | ON                               |

You can disable **NDB** transaction support by setting this variable’s values to **OFF** (not recommended). The default is **ON**.

- **ndb_version**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndb_version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
</tbody>
</table>
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
</table>

**NDB engine version, as a composite integer.**

- **ndb_version_string**
  
  **System Variable**  ndb_version_string  
  **Scope**  Global  
  **Dynamic**  No  
  **SET_VAR Hint Applies**  No  
  **Type**  String  
  **Default Value**  

**NDB engine version in ndb-x.y.z format.**

- **server_id_bits**
  
  **Command-Line Format**  --server-id-bits=#  
  **System Variable**  server_id_bits  
  **Scope**  Global  
  **Dynamic**  No  
  **SET_VAR Hint Applies**  No  
  **Type**  Integer  
  **Default Value**  32  
  **Minimum Value**  7  
  **Maximum Value**  32  

This variable indicates the number of least significant bits within the 32-bit server_id which actually identify the server. Indicating that the server is actually identified by fewer than 32 bits makes it possible for some of the remaining bits to be used for other purposes, such as storing user data generated by applications using the NDB API's Event API within the AnyValue of an OperationOptions structure (NDB Cluster uses the AnyValue to store the server ID).

When extracting the effective server ID from server_id for purposes such as detection of replication loops, the server ignores the remaining bits. The server_id_bits variable is used to mask out any irrelevant bits of server_id in the I/O and SQL threads when deciding whether an event should be ignored based on the server ID.

This data can be read from the binary log by mysqlbinlog, provided that it is run with its own server_id_bits variable set to 32 (the default).

If the value of server_id greater than or equal to 2 to the power of server_id_bits; otherwise, mysqld refuses to start.

This system variable is supported only by NDB Cluster. It is not supported in the standard MySQL 8.0 Server.

- **slave_allow_batching**
  
  **Command-Line Format**  --slave-allow-batching[={OFF|ON}]  
  **System Variable**  slave_allow_batching  
  **Scope**  Global  

Whether or not batched updates are enabled on NDB Cluster replicas.

Setting this variable has an effect only when using replication with the NDB storage engine; in MySQL Server 8.0, it is present but does nothing. For more information, see Section 8.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

• **transaction_allow_batching**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>transaction_allow_batching</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Session</td>
</tr>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>OFF</td>
</tr>
</tbody>
</table>

When set to 1 or ON, this variable enables batching of statements within the same transaction. To use this variable, autocommit must first be disabled by setting it to 0 or OFF; otherwise, setting transaction_allow_batching has no effect.

It is safe to use this variable with transactions that performs writes only, as having it enabled can lead to reads from the “before” image. You should ensure that any pending transactions are committed (using an explicit COMMIT if desired) before issuing a SELECT.

**Important**

transaction_allow_batching should not be used whenever there is the possibility that the effects of a given statement depend on the outcome of a previous statement within the same transaction.

This variable is currently supported for NDB Cluster only.

The system variables in the following list all relate to the ndbinfo information database.

• **ndbinfo_database**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndbinfo_database</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>ndbinfo</td>
</tr>
</tbody>
</table>

Shows the name used for the NDB information database; the default is ndbinfo. This is a read-only variable whose value is determined at compile time.

• **ndbinfo_max_bytes**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndbinfo-max-bytes=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>System Variable</td>
<td>ndbinfo_max_bytes</td>
</tr>
</tbody>
</table>
MySQL Server Options and Variables for NDB Cluster

<table>
<thead>
<tr>
<th>Scope</th>
<th>Global, Session</th>
</tr>
</thead>
<tbody>
<tr>
<td>Dynamic</td>
<td>Yes</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
</tbody>
</table>

Used in testing and debugging only.

- **ndbinfo_max_rows**
  
  Command-Line Format: `--ndbinfo-max-rows=#`
  
  System Variable: `ndbinfo_max_rows`
  
  Scope: Global, Session
  
  Dynamic: Yes
  
  SET_VAR Hint Applies: No
  
  Type: Integer
  
  Default Value: 10

Used in testing and debugging only.

- **ndbinfo_offline**
  
  System Variable: `ndbinfo_offline`
  
  Scope: Global
  
  Dynamic: Yes
  
  SET_VAR Hint Applies: No
  
  Type: Boolean
  
  Default Value: OFF

Place the `ndbinfo` database into offline mode, in which tables and views can be opened even when they do not actually exist, or when they exist but have different definitions in NDB. No rows are returned from such tables (or views).

- **ndbinfo_show_hidden**
  
  Command-Line Format: `--ndbinfo-show-hidden=[{OFF|ON}]`
  
  System Variable: `ndbinfo_show_hidden`
  
  Scope: Global, Session
  
  Dynamic: Yes
  
  SET_VAR Hint Applies: No
  
  Type: Boolean
  
  Default Value: OFF
  
  Valid Values:
  
  | OFF |
  | ON  |

Whether or not the `ndbinfo` database's underlying internal tables are shown in the `mysql` client. The default is OFF.
• **ndbinfo_table_prefix**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndbinfo_table_prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>ndb$</td>
</tr>
</tbody>
</table>

The prefix used in naming the ndbinfo database's base tables (normally hidden, unless exposed by setting `ndbinfo_show_hidden`). This is a read-only variable whose default value is `ndb$`; the prefix itself is determined at compile time.

• **ndbinfo_version**

<table>
<thead>
<tr>
<th>System Variable</th>
<th>ndbinfo_version</th>
</tr>
</thead>
<tbody>
<tr>
<td>Scope</td>
<td>Global</td>
</tr>
<tr>
<td>Dynamic</td>
<td>No</td>
</tr>
<tr>
<td>SET_VAR Hint Applies</td>
<td>No</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

Shows the version of the ndbinfo engine in use; read-only.

### 5.3.9.3 NDB Cluster Status Variables

This section provides detailed information about MySQL server status variables that relate to NDB Cluster and the NDB storage engine. For status variables not specific to NDB Cluster, and for general information on using status variables, see Server Status Variables.

• **Handler_discover**

The MySQL server can ask the NDBCLUSTER storage engine if it knows about a table with a given name. This is called discovery. `Handler_discover` indicates the number of times that tables have been discovered using this mechanism.

• **Ndb_api_adaptive_send_deferred_count**

Number of adaptive send calls that were not actually sent.

For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

• **Ndb_api_adaptive_send_deferred_count_session**

Number of adaptive send calls that were not actually sent.

For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

• **Ndb_api_adaptive_send_deferred_count_replica**

Number of adaptive send calls that were not actually sent by this replica.

For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

• **Ndb_api_adaptive_send_deferred_count_slave**
Number of adaptive send calls that were not actually sent by this replica.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count`
  Number of adaptive send calls using forced-send sent by this MySQL Server (SQL node).
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count_session`
  Number of adaptive send calls using forced-send sent in this client session.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count_replica`
  Number of adaptive send calls using forced-send sent by this replica.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_forced_count_slave`

  Number of adaptive send calls using forced-send sent by this replica.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count`
  Number of adaptive send calls without forced-send sent by this MySQL server (SQL node).
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count_session`
  Number of adaptive send calls without forced-send sent in this client session.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_adaptive_send_unforced_count_replica`
  Number of adaptive send calls without forced-send sent by this replica.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

Note

Deprecated in NDB 8.0.23; use `Ndb_api_adaptive_send_deferred_count_replica` instead.

Number of adaptive send calls that were not actually sent by this replica.
MySQL Server Options and Variables for NDB Cluster

• **Ndb_api_adaptive_send_unforced_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_adaptive_send_unforced_count_replica` instead.

  Number of adaptive send calls without forced-send sent by this replica.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_bytes_sent_count_session**

  Amount of data (in bytes) sent to the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_bytes_sent_count_replica**

  Amount of data (in bytes) sent to the data nodes by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_bytes_sent_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_bytes_sent_count_replica` instead.

  Amount of data (in bytes) sent to the data nodes by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_bytes_sent_count**

  Amount of data (in bytes) sent to the data nodes by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_bytes_received_count_session**

  Amount of data (in bytes) received from the data nodes in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”. 
• *Ndb_api_bytes_received_count_replica*

Amount of data (in bytes) received from the data nodes by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• *Ndb_api_bytes_received_count_slave*

Note

Deprecated in NDB 8.0.23; use `Ndb_api_bytes_received_count_replica` instead.

Amount of data (in bytes) received from the data nodes by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• *Ndb_api_bytes_received_count*

Amount of data (in bytes) received from the data nodes by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• *Ndb_api_event_data_count_injector*

The number of row change events received by the NDB binlog injector thread.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• *Ndb_api_event_data_count*

The number of row change events received by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• *Ndb_api_event_nondata_count_injector*

The number of events received, other than row change events, by the NDB binary log injector thread.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
• **Ndb_api_event_nondata_count**

  The number of events received, other than row change events, by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_event_bytes_count_injector**

  The number of bytes of events received by the NDB binlog injector thread.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_event_bytes_count**

  The number of bytes of events received by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_pk_op_count_session**

  The number of operations in this client session based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_pk_op_count_replica**

  The number of operations by this replica based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.


• **Ndb_api_pk_op_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_pk_op_count_replica` instead.

  The number of operations by this replica based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see [Section 7.13, “NDB API Statistics Counters and Variables”](#).

• **Ndb_api_pk_op_count**

  The number of operations by this MySQL Server (SQL node) based on or using primary keys. This includes operations on blob tables, implicit unlock operations, and auto-increment operations, as well as user-visible primary key operations.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see [Section 7.13, “NDB API Statistics Counters and Variables”](#).

• **Ndb_api_pruned_scan_count_session**

  The number of scans in this client session that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see [Section 7.13, “NDB API Statistics Counters and Variables”](#).

• **Ndb_api_pruned_scan_count_replica**

  The number of scans by this replica that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see [Section 7.13, “NDB API Statistics Counters and Variables”](#).

• **Ndb_api_pruned_scan_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_pruned_scan_count_replica` instead.

  The number of scans by this replica that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see [Section 7.13, “NDB API Statistics Counters and Variables”](#).
• **Ndb_api_pruned_scan_count**

  The number of scans by this MySQL Server (SQL node) that have been pruned to a single partition.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_range_scan_count_session**

  The number of range scans that have been started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_range_scan_count_replica**

  The number of range scans that have been started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_range_scan_count_slave**

  Note

  Deprecated in NDB 8.0.23; use `Ndb_api_range_scan_count_replica` instead.

  The number of range scans that have been started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_range_scan_count**

  The number of range scans that have been started by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_read_row_count_session**

  The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_read_row_count_replica**
MySQL Server Options and Variables for NDB Cluster

The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_read_row_count_slave**

  Note

  Deprecated in NDB 8.0.23; use `Ndb_api_read_row_count_replica` instead.

  The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_read_row_count**

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  You should be aware that this value may not be completely accurate with regard to rows read by `SELECT COUNT(*)` queries, due to the fact that, in this case, the MySQL server actually reads pseudo-rows in the form `[table fragment ID]:[number of rows in fragment]` and sums the rows per fragment for all fragments in the table to derive an estimated count for all rows. `Ndb_api_read_row_count` uses this estimate and not the actual number of rows in the table.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_scan_batch_count_session**

  The number of batches of rows received in this client session. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
• **Ndb_api_scan_batch_count_replica**

  The number of batches of rows received by this replica. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_scan_batch_count_slave**

  **Note**
  
  Deprecated in NDB 8.0.23; use `Ndb_api_scan_batch_count_replica` instead.

  The number of batches of rows received by this replica. 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_scan_batch_count**

  The number of batches of rows received by this MySQL Server (SQL node). 1 batch is defined as 1 set of scan results from a single fragment.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_table_scan_count_session**

  The number of table scans that have been started in this client session, including scans of internal tables.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_table_scan_count_replica**

  The number of table scans that have been started by this replica, including scans of internal tables.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
• **Ndb_api_table_scan_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_table_scan_count_replica` instead.

  The number of table scans that have been started by this replica, including scans of internal tables.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_table_scan_count**

  The number of table scans that have been started by this MySQL Server (SQL node), including scans of internal tables.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_abort_count_session**

  The number of transactions aborted in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_abort_count_replica**

  The number of transactions aborted by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_abort_count_slave**

  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_trans_abort_count_replica` instead.

  The number of transactions aborted by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
MySQL Server Options and Variables for NDB Cluster

- **Ndb_api_trans_abort_count**
  The number of transactions aborted by this MySQL Server (SQL node).
  
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.
  
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_trans_close_count_session**
  The number of transactions closed in this client session. This value may be greater than the sum of `Ndb_api_trans_commit_count_session` and `Ndb_api_trans_abort_count_session`, since some transactions may have been rolled back.
  
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.
  
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_trans_close_count_replica**
  The number of transactions closed by this replica. This value may be greater than the sum of `Ndb_api_trans_commit_count_replica` and `Ndb_api_trans_abort_count_replica`, since some transactions may have been rolled back.
  
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.
  
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_trans_close_count_slave**
  
  **Note**
  
  Deprecated in NDB 8.0.23; use `Ndb_api_trans_close_count_replica` instead.
  
  The number of transactions closed by this replica. This value may be greater than the sum of `Ndb_api_trans_commit_count_replica` and `Ndb_api_trans_abort_count_replica`, since some transactions may have been rolled back.
  
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.
  
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- **Ndb_api_trans_close_count**
  The number of transactions closed by this MySQL Server (SQL node). This value may be greater than the sum of `Ndb_api_trans_commit_count` and `Ndb_api_trans_abort_count`, since some transactions may have been rolled back.
  
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.
  
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

---

226
• **Ndb_api_trans_commit_count_session**

The number of transactions committed in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_commit_count_replica**

The number of transactions committed by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_commit_count_slave**

Note

Deprecated in NDB 8.0.23; use `Ndb_api_trans_commit_count_replica` instead.

The number of transactions committed by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_commit_count**

The number of transactions committed by this MySQL Server (SQL node).

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_local_read_row_count_session**

The total number of rows that have been read in this client session. This includes all rows read by any primary key, unique key, or scan operation made in this client session.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_local_read_row_count_replica**

The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.


MySQL Server Options and Variables for NDB Cluster

• **Ndb_api_trans_local_read_row_count_slave**

  Note
  Deprecated in NDB 8.0.23; use `Ndb_api_trans_local_read_row_count_replica` instead.

  The total number of rows that have been read by this replica. This includes all rows read by any primary key, unique key, or scan operation made by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_local_read_row_count**

  The total number of rows that have been read by this MySQL Server (SQL node). This includes all rows read by any primary key, unique key, or scan operation made by this MySQL Server (SQL node).

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_start_count_session**

  The number of transactions started in this client session.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_start_count_replica**

  The number of transactions started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_trans_start_count_slave**

  Note
  Deprecated in NDB 8.0.23; use `Ndb_api_trans_start_count_replica` instead.

  The number of transactions started by this replica.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”. 228
MySQL Server Options and Variables for NDB Cluster

• **Ndb_api_trans_start_count**
  The number of transactions started by this MySQL Server (SQL node).
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_uk_op_count_session**
  The number of operations in this client session based on or using unique keys.
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_uk_op_count_replica**
  The number of operations by this replica based on or using unique keys.
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_uk_op_count_slave**
  **Note**
  Deprecated in NDB 8.0.23; use `Ndb_api_uk_op_count_replica` instead.
  The number of operations by this replica based on or using unique keys.
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_uk_op_count**
  The number of operations by this MySQL Server (SQL node) based on or using unique keys.
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_wait_exec_complete_count_session**
  The number of times a thread has been blocked in this client session while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.
  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.
  For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• **Ndb_api_wait_exec_complete_count_replica**
The number of times a thread has been blocked by this replica while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count_slave`

Note

Deprecated in NDB 8.0.23; use `Ndb_api_wait_exec_complete_count_replica` instead.

The number of times a thread has been blocked by this replica while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_exec_complete_count`

The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for execution of an operation to complete. This includes all `execute()` calls as well as implicit executes for blob and auto-increment operations not visible to clients.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_session`

The number of times a thread has been blocked in this client session waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysqld`.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

- `Ndb_api_wait_meta_request_count_replica`

The number of times a thread has been blocked by this replica waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
• Ndb_api_wait_meta_request_count_slave

**Note**
Deprecated in NDB 8.0.23; use Ndb_api_wait_meta_request_count_replica instead.

The number of times a thread has been blocked by this replica waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either SHOW GLOBAL STATUS or SHOW SESSION STATUS, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• Ndb_api_wait_meta_request_count

The number of times a thread has been blocked by this MySQL Server (SQL node) waiting for a metadata-based signal, such as is expected for DDL requests, new epochs, and seizure of transaction records.

Although this variable can be read using either SHOW GLOBAL STATUS or SHOW SESSION STATUS, it is effectively global in scope.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• Ndb_api_wait_nanos_count_session

Total time (in nanoseconds) spent in this client session waiting for any type of signal from the data nodes.

Although this variable can be read using either SHOW GLOBAL STATUS or SHOW SESSION STATUS, it relates to the current session only, and is not affected by any other clients of this mysqld.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• Ndb_api_wait_nanos_count_replica

Total time (in nanoseconds) spent by this replica waiting for any type of signal from the data nodes.

Although this variable can be read using either SHOW GLOBAL STATUS or SHOW SESSION STATUS, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.

• Ndb_api_wait_nanos_count_slave

**Note**
Deprecated in NDB 8.0.23; use Ndb_api_wait_nanos_count_replica instead.

Total time (in nanoseconds) spent by this replica waiting for any type of signal from the data nodes.

Although this variable can be read using either SHOW GLOBAL STATUS or SHOW SESSION STATUS, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

For more information, see Section 7.13, “NDB API Statistics Counters and Variables”.
MySQL Server Options and Variables for NDB Cluster

- **Ndb_api_wait_nanos_count**

  Total time (in nanoseconds) spent by this MySQL Server (SQL node) waiting for any type of signal from the data nodes.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

- **Ndb_api_wait_scan_result_count_session**

  The number of times a thread has been blocked in this client session while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it relates to the current session only, and is not affected by any other clients of this `mysql`. For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

- **Ndb_api_wait_scan_result_count_replica**

  The number of times a thread has been blocked by this replica while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

- **Ndb_api_wait_scan_result_count_slave**

  Note

  Deprecated in NDB 8.0.23; use `Ndb_api_wait_scan_result_count_replica` instead.

  The number of times a thread has been blocked by this replica while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope. If this MySQL server does not act as a replica, or does not use NDB tables, this value is always 0.

  For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

- **Ndb_api_wait_scan_result_count**

  The number of times a thread has been blocked by this MySQL Server (SQL node) while waiting for a scan-based signal, such as when waiting for more results from a scan, or when waiting for a scan to close.

  Although this variable can be read using either `SHOW GLOBAL STATUS` or `SHOW SESSION STATUS`, it is effectively global in scope.

  For more information, see Section 7.13, "NDB API Statistics Counters and Variables".

- **Ndb_cluster_node_id**

  If the server is acting as an NDB Cluster node, then the value of this variable its node ID in the cluster.
MySQL Server Options and Variables for NDB Cluster

If the server is not part of an NDB Cluster, then the value of this variable is 0.

• **Ndb_config_from_host**

  If the server is part of an NDB Cluster, the value of this variable is the host name or IP address of the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is an empty string.

• **Ndb_config_from_port**

  If the server is part of an NDB Cluster, the value of this variable is the number of the port through which it is connected to the Cluster management server from which it gets its configuration data.

  If the server is not part of an NDB Cluster, then the value of this variable is 0.

• **Ndb_config_generation**

  Shows the generation number of the cluster's current configuration. This can be used as an indicator to determine whether the configuration of the cluster has changed since this SQL node last connected to the cluster.

• **Ndb_conflict_fn_max_del_win**

  Shows the number of times that a row was rejected on the current SQL node due to NDB Cluster Replication conflict resolution using `NDB$MAX_DELETE_WIN()`, since the last time that this `mysqld` was started.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **Ndb_conflict_fn_max**

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that this `mysqld` was started.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **Ndb_conflict_fn_old**

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **Ndb_conflict_fn_epoch**

  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **Ndb_conflict_fn_epoch2**

  Shows the number of rows found to be in conflict in NDB Cluster Replication conflict resolution, when using `NDB$EPOCH2()`, on the source designated as the primary since the last time it was restarted.

  For more information, see `NDB$EPOCH2()`.

• **Ndb_conflict_fn_epoch_trans**
Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS()` conflict resolution on a given `mysqld` since the last time it was restarted.

For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- **Ndb_conflict_fn_epoch2_trans**
  Used in NDB Cluster Replication conflict resolution, this variable shows the number of rows found to be in conflict using `NDB$EPOCH_TRANS2()` conflict resolution on a given `mysqld` since the last time it was restarted.

  For more information, see `NDB$EPOCH2_TRANS()`.

- **Ndb_conflict_last_conflict_epoch**
  The most recent epoch in which a conflict was detected on this replica. You can compare this value with `Ndb_replica_max_replicated_epoch`; if `Ndb_replica_max_replicated_epoch` is greater than `Ndb_conflict_last_conflict_epoch`, no conflicts have yet been detected.

  See Section 8.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- **Ndb_conflict_reflected_op_discard_count**
  When using NDB Cluster Replication conflict resolution, this is the number of reflected operations that were not applied on the secondary, due to encountering an error during execution.

  See Section 8.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- **Ndb_conflict_reflected_op_prepare_count**
  When using conflict resolution with NDB Cluster Replication, this status variable contains the number of reflected operations that have been defined (that is, prepared for execution on the secondary).

  See Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- **Ndb_conflict_refresh_op_count**
  When using conflict resolution with NDB Cluster Replication, this gives the number of refresh operations that have been prepared for execution on the secondary.

  See Section 8.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- **Ndb_conflict_last_stable_epoch**
  Number of rows found to be in conflict by a transactional conflict function

  See Section 8.11, “NDB Cluster Replication Conflict Resolution”, for more information.

- **Ndb_conflict_trans_row_conflict_count**
  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of rows found to be directly in-conflict by a transactional conflict function on a given `mysqld` since the last time it was restarted.

  Currently, the only transactional conflict detection function supported by NDB Cluster is `NDB$EPOCH_TRANS()`, so this status variable is effectively the same as `Ndb_conflict_fn_epoch_trans`.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- **Ndb_conflict_trans_row_reject_count**
Used in NDB Cluster Replication conflict resolution, this status variable shows the total number of rows realigned due to being determined as conflicting by a transactional conflict detection function. This includes not only \texttt{Ndb\_conflict\_trans\_row\_conflict\_count}, but any rows in or dependent on conflicting transactions.

For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- \texttt{Ndb\_conflict\_trans\_reject\_count}

  Used in NDB Cluster Replication conflict resolution, this status variable shows the number of transactions found to be in conflict by a transactional conflict detection function.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- \texttt{Ndb\_conflict\_trans\_detect\_iter\_count}

  Used in NDB Cluster Replication conflict resolution, this shows the number of internal iterations required to commit an epoch transaction. Should be (slightly) greater than or equal to \texttt{Ndb\_conflict\_trans\_commit\_count}.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- \texttt{Ndb\_conflict\_trans\_commit\_count}

  Used in NDB Cluster Replication conflict resolution, this shows the number of epoch transactions committed after they required transactional conflict handling.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

- \texttt{Ndb\_epoch\_delete\_delete\_count}

  When using delete-delete conflict detection, this is the number of delete-delete conflicts detected, where a delete operation is applied, but the indicated row does not exist.

- \texttt{Ndb\_execute\_count}

  Provides the number of round trips to the NDB kernel made by operations.

- \texttt{Ndb\_last\_commit\_epoch\_server}

  The epoch most recently committed by NDB.

- \texttt{Ndb\_last\_commit\_epoch\_session}

  The epoch most recently committed by this NDB client.

- \texttt{Ndb\_metadata\_detected\_count}

  The number of times since this server was last started that the NDB metadata change detection thread has discovered changes with respect to the MySQL data dictionary.

  Added in NDB 8.0.16.

- \texttt{Ndb\_metadata\_excluded\_count}

  The number of metadata objects that the NDB binlog thread has been unable to synchronize on this SQL node since it was last restarted.

  Should an object be excluded, it is not again considered for automatic synchronization until the user corrects the mismatch manually. This can be done by attempting to use the table with a statement such as \texttt{SHOW CREATE TABLE table, SELECT * FROM table}, or any other statement that would trigger table discovery.
Added in NDB 8.0.18. Prior to NDB 8.0.22, this variable was named `Ndb_metadata_blacklist_size`.

- **Ndb_metadata_synced_count**
  
The number of NDB metadata objects which have been synchronized on this SQL node since it was last restarted.
  
  Added in NDB 8.0.18.

- **Ndb_number_of_data_nodes**
  
  If the server is part of an NDB Cluster, the value of this variable is the number of data nodes in the cluster.
  
  If the server is not part of an NDB Cluster, then the value of this variable is 0.

- **Ndb_pushed_queries_defined**
  
The total number of joins pushed down to the NDB kernel for distributed handling on the data nodes.

  **Note**
  
  Joins tested using `EXPLAIN` that can be pushed down contribute to this number.

- **Ndb_pushed_queries_dropped**
  
The number of joins that were pushed down to the NDB kernel but that could not be handled there.

- **Ndb_pushed_queries_executed**
  
The number of joins successfully pushed down to NDB and executed there.

- **Ndb_pushed_reads**
  
The number of rows returned to `mysqld` from the NDB kernel by joins that were pushed down.

  **Note**
  
  Executing `EXPLAIN` on joins that can be pushed down to NDB does not add to this number.

- **Ndb_pruned_scan_count**
  
  This variable holds a count of the number of scans executed by `NDBCLUSTER` since the NDB Cluster was last started where `NDBCLUSTER` was able to use partition pruning.

  Using this variable together with `Ndb_scan_count` can be helpful in schema design to maximize the ability of the server to prune scans to a single table partition, thereby involving replica only a single data node.

- **Ndb_replica_max_replicated_epoch**
  
The most recently committed epoch on this replica. You can compare this value with `Ndb_conflict_last_conflict_epoch`; if `Ndb_replica_max_replicated_epoch` is the greater of the two, no conflicts have yet been detected.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

236
• **Ndb_scan_count**

This variable holds a count of the total number of scans executed by NDBCLUSTER since the NDB Cluster was last started.

• **Ndb_slave_max_replicated_epoch**

  **Note**
  
  Deprecated in NDB 8.0.23; use `Ndb_slave_max_replicated_epoch` instead.

  The most recently committed epoch on this replica. You can compare this value with `Ndb_conflict_last_conflict_epoch`; if `Ndb_slave_max_replicated_epoch` is the greater of the two, no conflicts have yet been detected.

  For more information, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

• **Ndb_system_name**

  If this MySQL Server is connected to an NDB cluster, this read-only variable shows the cluster system name. Otherwise, the value is an empty string.

• **Ndb_trans_hint_count_session**

  The number of transactions using hints that have been started in the current session. Compare with `Ndb_api_trans_start_count_session` to obtain the proportion of all NDB transactions able to use hints. Added in NDB 8.0.17.

### 5.3.10 NDB Cluster TCP/IP Connections

TCP/IP is the default transport mechanism for all connections between nodes in an NDB Cluster. Normally it is not necessary to define TCP/IP connections; NDB Cluster automatically sets up such connections for all data nodes, management nodes, and SQL or API nodes.

  **Note**
  
  For an exception to this rule, see Section 5.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections”.

To override the default connection parameters, it is necessary to define a connection using one or more `[tcp]` sections in the `config.ini` file. Each `[tcp]` section explicitly defines a TCP/IP connection between two NDB Cluster nodes, and must contain at a minimum the parameters `NodeId1` and `NodeId2`, as well as any connection parameters to override.

It is also possible to change the default values for these parameters by setting them in the `[tcp default]` section.

  **Important**
  
  Any `[tcp]` sections in the `config.ini` file should be listed last, following all other sections in the file. However, this is not required for a `[tcp default]` section. This requirement is a known issue with the way in which the `config.ini` file is read by the NDB Cluster management server.

Connection parameters which can be set in `[tcp]` and `[tcp default]` sections of the `config.ini` file are listed here:

• **AllowUnresolvedHostNames**

  By default, when a management node fails to resolve a host name while trying to connect, this results in a fatal error. This behavior can be overridden by setting `AllowUnresolvedHostNames` to
true in the [tcp default] section of the global configuration file (usually named config.ini), in which case failure to resolve a host name is treated as a warning and ndb_mgmd startup continues uninterrupted.

- **Checksum**

  This parameter is a boolean parameter (enabled by setting it to `Y` or `1`, disabled by setting it to `N` or `0`). It is disabled by default. When it is enabled, checksums for all messages are calculated before they placed in the send buffer. This feature ensures that messages are not corrupted while waiting in the send buffer, or by the transport mechanism.

- **Group**

  When `ndb_optimized_node_selection` is enabled, node proximity is used in some cases to select which node to connect to. This parameter can be used to influence proximity by setting it to a lower value, which is interpreted as “closer”. See the description of the system variable for more information.

- **HostName1**

  The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- **HostName2**

  The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given TCP connection between two nodes. The values used for these parameters can be host names or IP addresses.

- **NodeId1**

  To identify a connection between two nodes it is necessary to provide their node IDs in the [tcp] section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique Id values for each of these nodes as described in Section 5.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- **NodeId2**

  To identify a connection between two nodes it is necessary to provide their node IDs in the [tcp] section of the configuration file as the values of `NodeId1` and `NodeId2`. These are the same unique Id values for each of these nodes as described in Section 5.3.7, “Defining SQL and Other API Nodes in an NDB Cluster”.

- **OverloadLimit**

  When more than this many unsent bytes are in the send buffer, the connection is considered overloaded.

  This parameter can be used to determine the amount of unsent data that must be present in the send buffer before the connection is considered overloaded. See Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”, for more information.
**PreSendChecksum**

If this parameter and Checksum are both enabled, perform pre-send checksum checks, and check all TCP signals between nodes for errors. Has no effect if Checksum is not also enabled.

**ReceiveBufferMemory**

Specifies the size of the buffer used when receiving data from the TCP/IP socket.

The default value of this parameter is 2MB. The minimum possible value is 16KB; the theoretical maximum is 4GB.

**SendBufferMemory**

TCP transporters use a buffer to store all messages before performing the send call to the operating system. When this buffer reaches 64KB its contents are sent; these are also sent when a round of messages have been executed. To handle temporary overload situations it is also possible to define a bigger send buffer.

If this parameter is set explicitly, then the memory is not dedicated to each transporter; instead, the value used denotes the hard limit for how much memory (out of the total available memory—that is, TotalSendBufferMemory) that may be used by a single transporter. For more information about configuring dynamic transporter send buffer memory allocation in NDB Cluster, see Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

The default size of the send buffer is 2MB, which is the size recommended in most situations. The minimum size is 64 KB; the theoretical maximum is 4 GB.

**SendSignalId**

To be able to retrace a distributed message datagram, it is necessary to identify each message. When this parameter is set to Y, message IDs are transported over the network. This feature is disabled by default in production builds, and enabled in -debug builds.

**TcpBind_INADDR_ANY**

Setting this parameter to TRUE or 1 binds IP_ADDR_ANY so that connections can be made from anywhere (for autogenerated connections). The default is FALSE (0).

**TcpSpinTime**

Controls spin for a TCP transporter; no enable, set to a nonzero value. This works for both the data node and management or SQL node side of the connection.

**TCP_MAXSEG_SIZE**

Determines the size of the memory set during TCP transporter initialization. The default is recommended for most common usage cases.

**TCP_RCV_BUF_SIZE**

Determines the size of the receive buffer set during TCP transporter initialization. The default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

**TCP_SND_BUF_SIZE**

Determines the size of the receive buffer set during TCP transporter initialization. The default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.
Determines the size of the send buffer set during TCP transporter initialization. The default and minimum value is 0, which allows the operating system or platform to set this value. The default is recommended for most common usage cases.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, &quot;Performing a Rolling Restart of an NDB Cluster&quot;)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the --initial option</td>
</tr>
</tbody>
</table>

### 5.3.11 NDB Cluster TCP/IP Connections Using Direct Connections

Setting up a cluster using direct connections between data nodes requires specifying explicitly the crossover IP addresses of the data nodes so connected in the [tcp] section of the cluster config.ini file.

In the following example, we envision a cluster with at least four hosts, one each for a management server, an SQL node, and two data nodes. The cluster as a whole resides on the 172.23.72.* subnet of a LAN. In addition to the usual network connections, the two data nodes are connected directly using a standard crossover cable, and communicate with one another directly using IP addresses in the 1.1.0.* address range as shown:

```ini
[ndb_mgmd]
Id=1
HostName=172.23.72.20

[mysqld]
Id=2
HostName=172.23.72.21

[ndbd]
Id=3
HostName=172.23.72.22
[ndbd]
Id=4
HostName=172.23.72.23

[tcp]
NodeId1=3
NodeId2=4
HostName1=1.1.0.1
HostName2=1.1.0.2
```

The HostName1 and HostName2 parameters are used only when specifying direct connections.

The use of direct TCP connections between data nodes can improve the cluster's overall efficiency by enabling the data nodes to bypass an Ethernet device such as a switch, hub, or router, thus cutting down on the cluster's latency.

---

**Note**

To take the best advantage of direct connections in this fashion with more than two data nodes, you must have a direct connection between each data node and every other data node in the same node group.
Communications between NDB cluster nodes are normally handled using TCP/IP. The shared memory (SHM) transporter is distinguished by the fact that signals are transmitted by writing in memory rather than on a socket. The shared-memory transporter (SHM) can improve performance by negating up to 20% of the overhead required by a TCP connection when running an API node (usually an SQL node) and a data node together on the same host. You can enable a shared memory connection in either of the two ways listed here:

- By setting the UseShm data node configuration parameter to 1, and setting HostName for the data node and HostName for the API node to the same value.

- By using [shm] sections in the cluster configuration file, each containing settings for NodeId1 and NodeId2. This method is described in more detail later in this section.

Suppose a cluster is running a data node which has node ID 1 and an SQL node having node ID 51 on the same host computer at 10.0.0.1. To enable an SHM connection between these two nodes, all that is necessary is to insure that the following entries are included in the cluster configuration file:

```
[ndbd]
NodeId=1
HostName=10.0.0.1
UseShm=1

[mysqld]
NodeId=51
HostName=10.0.0.1
```

Important

The two entries just shown are in addition to any other entries and parameter settings needed by the cluster. A more complete example is shown later in this section.

Before starting data nodes that use SHM connections, it is also necessary to make sure that the operating system on each computer hosting such a data node has sufficient memory allocated to shared memory segments. See the documentation for your operating platform for information regarding this. In setups where multiple hosts are each running a data node and an API node, it is possible to enable shared memory on all such hosts by setting UseShm in the [ndbd default] section of the configuration file. This is shown in the example later in this section.

While not strictly required, tuning for all SHM connections in the cluster can be done by setting one or more of the following parameters in the [shm default] section of the cluster configuration (config.ini) file:

- **ShmSize**: Shared memory size
- **ShmSpinTime**: Time in µs to spin before sleeping
- **SendBufferMemory**: Size of buffer for signals sent from this node, in bytes.
- **SendSignalId**: Indicates that a signal ID is included in each signal sent through the transporter.
- **Checksum**: Indicates that a checksum is included in each signal sent through the transporter.
- **PreSendChecksum**: Checks of the checksum are made prior to sending the signal; Checksum must also be enabled for this to work

This example shows a simple setup with SHM connections defined on multiple hosts, in an NDB Cluster using 3 computers listed here by host name, hosting the node types shown:

1. **10.0.0.0**: The management server
2. **10.0.0.1**: A data node and an SQL node
3. **10.0.0.2**: A data node and an SQL node

In this scenario, each data node communicates with both the management server and the other data node using TCP transporters; each SQL node uses a shared memory transporter to communicate with the data nodes that is local to it, and a TCP transporter to communicate with the remote data node. A basic configuration reflecting this setup is enabled by the config.ini file whose contents are shown here:

```ini
[ndbd default]
DataDir=/path/to/datadir
UseShm=1
[shm default]
ShmSize=8M
ShmSpinTime=200
SendBufferMemory=4M
[tcp default]
SendBufferMemory=8M
[ndb_mgmd]
NodeId=49
Hostname=10.0.0.0
DataDir=/path/to/datadir
[ndbd]
NodeId=1
Hostname=10.0.0.1
DataDir=/path/to/datadir
[ndbd]
NodeId=2
Hostname=10.0.0.2
DataDir=/path/to/datadir
[mysqld]
NodeId=51
Hostname=10.0.0.1
[mysqld]
NodeId=52
Hostname=10.0.0.2
[api]
[api]
```

Parameters affecting all shared memory transporters are set in the `[shm default]` section; these can be overridden on a per-connection basis in one or more `[shm]` sections. Each such section must be associated with a given SHM connection using `NodeId1` and `NodeId2`; the values required for these parameters are the node IDs of the two nodes connected by the transporter. You can also identify the nodes by host name using `HostName1` and `HostName2`, but these parameters are not required.

The API nodes for which no host names are set use the TCP transporter to communicate with data nodes independent of the hosts on which they are started; the parameters and values set in the `[tcp default]` section of the configuration file apply to all TCP transporters in the cluster.

For optimum performance, you can define a spin time for the SHM transporter (`ShmSpinTime` parameter); this affects both the data node receiver thread and the poll owner (receive thread or user thread) in `NDB`.

- **Checksum**

  This parameter is a boolean (`Y/N`) parameter which is disabled by default. When it is enabled, checksums for all messages are calculated before being placed in the send buffer.

  This feature prevents messages from being corrupted while waiting in the send buffer. It also serves as a check against data being corrupted during transport.

- **Group**

  Determines the group proximity; a smaller value is interpreted as being closer. The default value is sufficient for most conditions.
NDB Cluster Shared-Memory Connections

- **HostName1**

  The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- **HostName2**

  The `HostName1` and `HostName2` parameters can be used to specify specific network interfaces to be used for a given SHM connection between two nodes. The values used for these parameters can be host names or IP addresses.

- **NodeId1**

  To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- **NodeId2**

  To identify a connection between two nodes it is necessary to provide node identifiers for each of them, as `NodeId1` and `NodeId2`.

- **NodeIdServer**

  Identify the server end of a shared memory connection. By default, this is the node ID of the data node.

- **OverloadLimit**

  When more than this many unsent bytes are in the send buffer, the connection is considered overloaded. See Section 5.3.14, "Configuring NDB Cluster Send Buffer Parameters", and Section 7.14.55, "The ndbinfo transporters Table", for more information.

- **PreSendChecksum**

  If this parameter and `Checksum` are both enabled, perform pre-send checksum checks, and check all SHM signals between nodes for errors. Has no effect if `Checksum` is not also enabled.

- **SendBufferMemory**

  Size (in bytes) of the shared memory buffer for signals sent from this node using a shared memory connection.

- **SendSignalId**

  To retrace the path of a distributed message, it is necessary to provide each message with a unique identifier. Setting this parameter to `Y` causes these message IDs to be transported over the network as well. This feature is disabled by default in production builds, and enabled in `-debug` builds.

- **ShmKey**

  When setting up shared memory segments, a node ID, expressed as an integer, is used to identify uniquely the shared memory segment to use for the communication. There is no default value. If `UseShm` is enabled, the shared memory key is calculated automatically by NDB.
Data Node Memory Management

- **ShmSize**

  Each SHM connection has a shared memory segment where messages between nodes are placed by the sender and read by the reader. The size of this segment is defined by `ShmSize`. The default value is 4MB.

- **ShmSpinTime**

  When receiving, the time to wait before sleeping, in microseconds.

- **SigNum**

  This parameter was used formerly to override operating system signal numbers; in NDB 8.0, it is no longer used, and any setting for it is ignored.

**Restart types.** Information about the restart types used by the parameter descriptions in this section is shown in the following table:

Table 5.15 NDB Cluster restart types

<table>
<thead>
<tr>
<th>Symbol</th>
<th>Restart Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>N</td>
<td>Node</td>
<td>The parameter can be updated using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”)</td>
</tr>
<tr>
<td>S</td>
<td>System</td>
<td>All cluster nodes must be shut down completely, then restarted, to effect a change in this parameter</td>
</tr>
<tr>
<td>I</td>
<td>Initial</td>
<td>Data nodes must be restarted using the <code>--initial</code> option</td>
</tr>
</tbody>
</table>

**5.3.13 Data Node Memory Management**

All memory allocation for a data node is performed when the node is started. This ensures that the data node can run in a stable manner without using swap memory, so that NDB can be used for latency-sensitive (realtime) applications. The following types of memory are allocated on data node startup:

- Data memory
- Shared global memory
- Redo log buffers
- Job buffers
- Send buffers
- Page cache for disk data records
- Schema transaction memory
- Transaction memory
- Undo log buffer
- Query memory
- Block objects
- Schema memory
Data Node Memory Management

- Block data structures
- Long signal memory
- Shared memory communication buffers

The NDB memory manager, which regulates most data node memory, handles the following memory resources:

- **Data Memory** *(DataMemory)*
- **Redo log buffers** *(RedoBuffer)*
- **Job buffers**
- **Send buffers** *(SendBufferMemory, TotalSendBufferMemory, ExtraSendBufferMemory, ReservedSendBufferMemory)*
- **Disk Data record page cache** *(DiskPageBufferMemory, DiskPageBufferEntries)*
- **Transaction memory** *(TransactionMemory)*
- **Query memory**
- **Disk access records**
- **File buffers**

Each of these resources is set up with a reserved memory area and a maximum memory area. The reserved memory area can be used only by the resource for which it is reserved and cannot be shared with other resources; a given resource can never allocate more than the maximum memory allowed for the resource. A resource that has no maximum memory can expand to use all the shared memory in the memory manager.

The size of the global shared memory for these resources is controlled by the *SharedGlobalMemory* configuration parameter (default: 128 MB).

Data memory is always reserved and never acquires any memory from shared memory. It is controlled using the *DataMemory* configuration parameter, whose maximum is 16384 GB. *DataMemory* is where records are stored, including hash indexes (approximately 15 bytes per row), ordered indexes (10-12 bytes per row per index), and row headers (16-32 bytes per row).

Redo log buffers also use reserved memory only; this is controlled by the *RedoBuffer* configuration parameter, which sets the size of the redo log buffer per LDM thread. This means that the actual amount of memory used is the value of this parameter multiplied by the number of LDM threads in the data node.

Job buffers use reserved memory only; the size of this memory is calculated by NDB, based on the numbers of threads of various types.

Send buffers have a reserved part but can also allocate an additional 25% of shared global memory. The send buffer reserved size is calculated in two steps:

1. Use the value of the *TotalSendBufferMemory* configuration parameter (no default value) or the sum of the individual send buffers used by all individual connections to the data node. A data node is connected to all other data nodes, to all API nodes, and to all management nodes. This means that, in a cluster with 2 data nodes, 2 management nodes, and 10 API nodes each data node has 13 node connections. Since the default value for *SendBufferMemory* for a data node connection is 2 MByte, this works out to 26 MB total.

2. To obtain the total reserved size for the send buffer, the value of the *ExtraSendBufferMemory* configuration parameter, if any (default value 0), is added to the value obtained in the previous step.
In other words, if \( \text{TotalSendBufferMemory} \) has been set, the send buffer size is 
\( \text{TotalSendBufferMemory} + \text{ExtraSendBufferMemory} \); otherwise, the size of the 
send buffer is equal to 
\( \left( \text{number of node connections} \right) \times \text{SendBufferMemory} + \text{ExtraSendBufferMemory} \).

The page cache for disk data records uses a reserved resource only; the size of this resource 
is controlled by the \( \text{DiskPageBufferMemory} \) configuration parameter (default 64 MB). 
Memory for 32 KB disk page entries is also allocated; the number of these is determined by the 
\( \text{DiskPageBufferEntries} \) configuration parameter (default 10).

Transaction memory has a reserved part that either is calculated by \( \text{NDB} \), or is set explicitly using the 
\( \text{TransactionMemory} \) configuration parameter introduced in NDB 8.0.19 (in previous releases, this 
value was always calculated by \( \text{NDB} \)); transaction memory can also use an unlimited amount of shared 
global memory. Transaction memory is used for all operational resources handling transactions, scans, 
locks, scan buffers, and trigger operations. It also holds table rows as they are updated, before the next 
commit writes them to data memory.

In NDB 8.0.16 and earlier, operational records used dedicated resources whose sizes were controlled 
by a number of configuration parameters. Beginning with NDB 8.0.17, these are all allocated from a 
common transaction memory resource and can also use resources from global shared memory. In 
NDB 8.0.19 and later, the size of this resource can be controlled using a single \( \text{TransactionMemory} \) 
configuration parameter.

Reserved memory for undo log buffers can be set using the \( \text{InitialLogFileGroup} \) configuration 
parameter. If an undo log buffer is created as part of a \text{CREATE LOGFILE GROUP} SQL statement, the 
memory is taken from the transaction memory.

A number of resources relating to metadata for Disk Data resources also have no reserved part, and 
use shared global memory only. Shared global shared memory is thus shared between send buffers, 
transaction memory, and Disk Data metadata.

If \( \text{TransactionMemory} \) is not set, it is calculated based on the following parameters:

- \( \text{MaxNoOfConcurrentOperations} \)
- \( \text{MaxNoOfConcurrentTransactions} \)
- \( \text{MaxNoOfFiredTriggers} \)
- \( \text{MaxNoOfLocalOperations} \)
- \( \text{MaxNoOfConcurrentIndexOperations} \)
- \( \text{MaxNoOfConcurrentScans} \)
- \( \text{MaxNoOfLocalScans} \)
- \( \text{BatchSizePerLocalScan} \)
- \( \text{TransactionBufferMemory} \)

When \( \text{TransactionMemory} \) is set explicitly, none of the configuration parameters just listed are used 
to calculate memory size. In addition, the parameters \( \text{MaxNoOfConcurrentIndexOperations} \), 
\( \text{MaxNoOfFiredTriggers} \), \( \text{MaxNoOfLocalOperations} \), and \( \text{MaxNoOfLocalScans} \) 
are incompatible with \( \text{TransactionMemory} \) and cannot be set concurrently with it; if 
\( \text{TransactionMemory} \) is set and any of these four parameters are also set in the \text{config.ini} 
configuration file, the management server cannot start. These four parameters are deprecated in NDB 
8.0.19; expect them to be removed from a future release of MySQL NDB Cluster.

The transaction memory resource contains a large number of memory pools. Each memory pool 
represents an object type and contains a set of objects; each pool includes a reserved part allocated to 
the pool at startup; this reserved memory is never returned to shared global memory. Reserved records
Data Node Memory Management

are found using a data structure having only a single level for fast retrieval, which means that a number of records in each pool should be reserved. The number of reserved records in each pool has some impact on performance and reserved memory allocation, but is generally necessary only in certain very advanced use cases to set the reserved sizes explicitly.

The size of the reserved part of the pool can be controlled by setting the following configuration parameters:

- `ReservedConcurrentIndexOperations`
- `ReservedFiredTriggers`
- `ReservedConcurrentOperations`
- `ReservedLocalScans`
- `ReservedConcurrentTransactions`
- `ReservedConcurrentScans`
- `ReservedTransactionBufferMemory`

If the parameters just listed are not set, the reserved setting is 25% of transaction memory. The number of reserved records is per data node; these records are split among the threads handling them (LDM and TC threads) on each node. In most cases, it is sufficient to set `TransactionMemory` alone, and to allow the number of records in pools to be governed by its value.

`MaxNoOfConcurrentScans` limits the number of concurrent scans that can be active in each TC thread. This is important in guarding against cluster overload.

`MaxNoOfConcurrentOperations` limits the number of operations that can be active at any one time in updating transactions. (Simple reads are not affected by this parameter.) This number needs to be limited because it is necessary to preallocate memory for node failure handling, and a resource must be available for handling the maximum number of active operations in one TC thread when commtending with node failures. It is imperative that `MaxNoOfConcurrentOperations` be set to the same number on all nodes (this can be done most easily by setting a value for it once, in the `[ndbd default]` section of the `config.ini` global configuration file). While its value can be increased using a rolling restart (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”), decreasing it in this way is not considered safe due to the possibility of a node failure occurring during the rolling restart.

It is possible to limit the size of a single transaction in NDB Cluster through the `MaxDMLOperationsPerTransaction` parameter. If this is not set, the size of one transaction is limited by `MaxNoOfConcurrentOperations` since this parameter limits the total number of concurrent operations per TC thread.

Schema memory size is controlled by the following set of configuration parameters:

- `MaxNoOfSubscriptions`
- `MaxNoOfSubscribers`
- `MaxNoOfConcurrentSubOperations`
- `MaxNoOfAttributes`
- `MaxNoOfTables`
- `MaxNoOfOrderedIndexes`
- `MaxNoOfUniqueHashIndexes`
- `MaxNoOfTriggers`
Configuring NDB Cluster Send Buffer Parameters

The number of nodes and the number of LDM threads also have a major impact on the size of schema memory since the number of partitions in each table and each partition (and its fragment replicas) have to be represented in schema memory.

In addition, a number of other records are allocated during startup. These are relatively small. Each block in each thread contains block objects that use memory. This memory size is also normally quite small compared to the other data node memory structures.

5.3.14 Configuring NDB Cluster Send Buffer Parameters

The NDB kernel employs a unified send buffer whose memory is allocated dynamically from a pool shared by all transporters. This means that the size of the send buffer can be adjusted as necessary. Configuration of the unified send buffer can accomplished by setting the following parameters:

- **TotalSendBufferMemory.** This parameter can be set for all types of NDB Cluster nodes—that is, it can be set in the [ndbd], [mgm], and [api] (or [mysql]) sections of the config.ini file. It represents the total amount of memory (in bytes) to be allocated by each node for which it is set for use among all configured transporters. If set, its minimum is 256KB; the maximum is 4294967039. To be backward-compatible with existing configurations, this parameter takes as its default value the sum of the maximum send buffer sizes of all configured transporters, plus an additional 32KB (one page) per transporter. The maximum depends on the type of transporter, as shown in the following table:

<table>
<thead>
<tr>
<th>Transporter</th>
<th>Maximum Send Buffer Size (bytes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>SendBufferMemory (default = 2M)</td>
</tr>
<tr>
<td>SHM</td>
<td>20K</td>
</tr>
</tbody>
</table>

This enables existing configurations to function in close to the same way as they did with NDB Cluster 6.3 and earlier, with the same amount of memory and send buffer space available to each transporter. However, memory that is unused by one transporter is not available to other transporters.

- **OverloadLimit.** This parameter is used in the config.ini file [tcp] section, and denotes the amount of unsent data (in bytes) that must be present in the send buffer before the connection is considered overloaded. When such an overload condition occurs, transactions that affect the overloaded connection fail with NDB API Error 1218 (*Send Buffers overloaded in NDB kernel*) until the overload status passes. The default value is 0, in which case the effective overload limit is calculated as $\text{SendBufferMemory} \times 0.8$ for a given connection. The maximum value for this parameter is 4G.

- **SendBufferMemory.** This value denotes a hard limit for the amount of memory that may be used by a single transporter out of the entire pool specified by TotalSendBufferMemory. However, the sum of SendBufferMemory for all configured transporters may be greater than the TotalSendBufferMemory that is set for a given node. This is a way to save memory when many nodes are in use, as long as the maximum amount of memory is never required by all transporters at the same time.

You can use the ndbinfo.transporters table to monitor send buffer memory usage, and to detect slowdown and overload conditions that can adversely affect performance.

5.4 Using High-Speed Interconnects with NDB Cluster

Even before design of NDBCLUSTER began in 1996, it was evident that one of the major problems to be encountered in building parallel databases would be communication between the nodes in the network. For this reason, NDBCLUSTER was designed from the very beginning to permit the use of a number of different data transport mechanisms. In this Manual, we use the term transporter for these.
The NDB Cluster codebase provides for four different transporters:

- **TCP/IP using 100 Mbps or gigabit Ethernet**, as discussed in Section 5.3.10, “NDB Cluster TCP/IP Connections”.

- **Direct (machine-to-machine) TCP/IP**: although this transporter uses the same TCP/IP protocol as mentioned in the previous item, it requires setting up the hardware differently and is configured differently as well. For this reason, it is considered a separate transport mechanism for NDB Cluster. See Section 5.3.11, “NDB Cluster TCP/IP Connections Using Direct Connections”, for details.

- **Shared memory (SHM)**. For more information about SHM, see Section 5.3.12, “NDB Cluster Shared-Memory Connections”.

- **Scalable Coherent Interface (SCI)**.

  **Note**

  Using SCI transporters in NDB Cluster requires specialized hardware, software, and MySQL binaries not available with NDB 8.0.

Most users today employ TCP/IP over Ethernet because it is ubiquitous. TCP/IP is also by far the best-tested transporter for use with NDB Cluster.

Regardless of the transporter used, NDB attempts to make sure that communication with data node processes is done using chunks that are as large as possible since this benefits all types of data transmission.
Chapter 6 NDB Cluster Programs

Table of Contents

6.1 ndbd — The NDB Cluster Data Node Daemon ................................................................. 252
6.2 ndbinfo_select_all — Select From ndbinfo Tables .......................................................... 259
6.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded) ...................................... 260
6.4 ndb_mgm — The NDB Cluster Management Server Daemon ............................................... 261
6.5 ndb_mgm — The NDB Cluster Management Client ............................................................ 269
6.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables .......... 272
6.7 ndb_config — Extract NDB Cluster Configuration Information ............................................. 274
6.8 ndb_delete_all — Delete All Rows from an NDB Table ...................................................... 283
6.9 ndb_desc — Describe NDB Tables ...................................................................................... 284
6.10 ndb_drop_index — Drop Index from an NDB Table ............................................................ 289
6.11 ndb_drop_table — Drop an NDB Table ............................................................................ 290
6.12 ndb_error_reporter — NDB Error-Reporting Utility ......................................................... 291
6.13 ndb_import — Import CSV Data Into NDB ...................................................................... 292
6.14 ndb_index_stat — NDB Index Statistics Utility ................................................................. 304
6.15 ndb_move_data — NDB Data Copy Utility ..................................................................... 309
6.16 ndb_perror — Obtain NDB Error Message Information ...................................................... 312
6.17 ndb_print_backup_file — Print NDB Backup File Contents .............................................. 313
6.18 ndb_print_file — Print NDB Disk Data File Contents ....................................................... 317
6.19 ndb_print_frag_file — Print NDB Fragment List File Contents ........................................ 318
6.20 ndb_print_schema_file — Print NDB Schema File Contents ............................................ 319
6.21 ndb_print_sys_file — Print NDB System File Contents ................................................... 319
6.22 ndb_redo_log_reader — Check and Print Content of Cluster Redo Log ......................... 319
6.23 ndb_restore — Restore an NDB Cluster Backup .............................................................. 322
  6.23.1 Restoring an NDB Backup to a Different Version of NDB Cluster ................................. 345
  6.23.2 Restoring to a different number of data nodes ............................................................... 346
  6.23.3 Restoring from a backup taken in parallel ................................................................... 349
6.24 ndb_select_all — Print Rows from an NDB Table ............................................................ 350
6.25 ndb_select_count — Print Row Counts for NDB Tables .................................................... 353
6.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED) .... 354
6.27 ndb_show_tables — Display List of NDB Tables ............................................................. 357
6.28 ndb_size.pl — NDBCLUSTER Size Requirement Estimator .............................................. 359
6.29 ndb_top — View CPU usage information for NDB threads .............................................. 361
6.30 ndb_waiter — Wait for NDB Cluster to Reach a Given Status .......................................... 366
6.31 ndbxfrm — Compress, Decompress, Encrypt, and Decrypt Files Created by NDB Cluster .... 368
6.32 Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs ... 371

Using and managing an NDB Cluster requires several specialized programs, which we describe in this chapter. We discuss the purposes of these programs in an NDB Cluster, how to use the programs, and what startup options are available for each of them.

These programs include the NDB Cluster data, management, and SQL node processes (ndbd, ndbmtd, ndb_mgm, and mysqld) and the management client (ndb_mgm).

Information about the program ndb_setup.py (now deprecated), used to start the NDB Cluster Auto-Installer, is also included in this section. You should be aware that Section 6.26, "ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)", contains information about the command-line client only; for information about using the GUI installer spawned by this program to configure and deploy an NDB Cluster, see The NDB Cluster Auto-Installer (NDB 7.5) (No longer supported).

For information about using mysqld as an NDB Cluster process, see Section 7.9, "MySQL Server Usage for NDB Cluster".

251
Other NDB utility, diagnostic, and example programs are included with the NDB Cluster distribution. These include `ndb_restore`, `ndb_show_tables`, and `ndb_config`. These programs are also covered in this section.

The final portion of this section contains tables of options that are common to all the various NDB Cluster programs.

### 6.1 ndbd — The NDB Cluster Data Node Daemon

`ndbd` is the process that is used to handle all the data in tables using the NDB Cluster storage engine. This is the process that empowers a data node to accomplish distributed transaction handling, node recovery, checkpointing to disk, online backup, and related tasks.

In an NDB Cluster, a set of `ndbd` processes cooperate in handling data. These processes can execute on the same computer (host) or on different computers. The correspondences between data nodes and Cluster hosts is completely configurable.

The following table includes command options specific to the NDB Cluster data node program `ndbd`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndbd`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--bind-address=name</code></td>
<td>Local bind address</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connect-delay=#</code></td>
<td>Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connect-retries=#</code></td>
<td>Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connect-retry-delay=#</code></td>
<td>Time to wait between attempts to contact a management server, in seconds; 0 means do not wait between attempts</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--daemon, -d</code></td>
<td>Start ndbd as daemon (default); override with --nodaemon</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--foreground</code></td>
<td>Run ndbd in foreground, provided for debugging purposes (implies --nodaemon)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--initial</code></td>
<td>Perform initial start of ndbd, including file system cleanup; consult documentation before using this option</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--initial-start</code></td>
<td>Perform partial initial start (requires --nowait-nodes)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--install[=name]</code></td>
<td>Used to install data node process as Windows service; does not apply on other platforms</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### ndbd — The NDB Cluster Data Node Daemon

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--logbuffer-size=#</td>
<td>Control size of log buffer; for use when debugging with many log messages being generated; default is sufficient for normal operations</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nodaemon</td>
<td>Do not start ndbd as daemon; provided for testing purposes</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nostart, -n</td>
<td>Do not start ndbd immediately; ndbd waits for command to start from ndb_mgm</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nowait-nodes=list</td>
<td>Do not wait for these data nodes to start (takes comma-separated list of node IDs); requires --ndb-nodeid</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--remove [=name]</td>
<td>Used to remove data node process that was previously installed as Windows service; does not apply on other platforms</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Write extra debugging information to node log</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

**Note**

All of these options also apply to the multithreaded version of this program (ndbmtd) and you may substitute “ndbmtd” for “ndbd” wherever the latter occurs in this section.

- **--bind-address**

  **Command-Line Format**  
  --bind-address=name

  **Type**  
  String

  **Default Value**  
  Causes ndbd to bind to a specific network interface (host name or IP address). This option has no default value.

- **--connect-delay=#**

  **Command-Line Format**  
  --connect-delay=#

  **Deprecated**  
  Yes

  **Type**  
  Numeric

  **Default Value**  
  5

  **Minimum Value**  
  0

  **Maximum Value**  
  3600

  Determines the time to wait between attempts to contact a management server when starting (the number of attempts is controlled by the --connect-retries option). The default is 5 seconds.

  This option is deprecated, and is subject to removal in a future release of NDB Cluster. Use --connect-retry-delay instead.
• --connect-retries=#

Command-Line Format: --connect-retries=#
Type: Numeric
Default Value: 12
Minimum Value: 0
Maximum Value: 65535

Set the number of times to retry a connection before giving up; 0 means 1 attempt only (and no retries). The default is 12 attempts. The time to wait between attempts is controlled by the --connect-retry-delay option.

• --connect-retry-delay=#

Command-Line Format: --connect-retry-delay=#
Type: Numeric
Default Value: 5
Minimum Value: 0
Maximum Value: 4294967295

Determines the time to wait between attempts to contact a management server when starting (the time between attempts is controlled by the --connect-retries option). The default is 5 seconds.

This option takes the place of the --connect-delay option, which is now deprecated and subject to removal in a future release of NDB Cluster.

• --daemon, -d

Command-Line Format: --daemon
Type: Boolean
Default Value: TRUE

Instructs ndbd or ndbmtd to execute as a daemon process. This is the default behavior. --nodaemon can be used to prevent the process from running as a daemon.

This option has no effect when running ndbd or ndbmtd on Windows platforms.

• --foreground

Command-Line Format: --foreground
Type: Boolean
Default Value: FALSE

Causes ndbd or ndbmtd to execute as a foreground process, primarily for debugging purposes. This option implies the --nodaemon option.

This option has no effect when running ndbd or ndbmtd on Windows platforms.

• --initial
ndbd — The NDB Cluster Data Node Daemon

**Default Value**  
FALSE

Instructs ndbd to perform an initial start. An initial start erases any files created for recovery purposes by earlier instances of ndbd. It also re-creates recovery log files. On some operating systems, this process can take a substantial amount of time.

An *--initial* start is to be used *only* when starting the ndbd process under very special circumstances; this is because this option causes all files to be removed from the NDB Cluster file system and all redo log files to be re-created. These circumstances are listed here:

- When performing a software upgrade which has changed the contents of any files.
- When restarting the node with a new version of ndbd.
- As a measure of last resort when for some reason the node restart or system restart repeatedly fails. In this case, be aware that this node can no longer be used to restore data due to the destruction of the data files.

**Warning**

To avoid the possibility of eventual data loss, it is recommended that you *not* use the *--initial* option together with *StopOnError = 0*. Instead, set *StopOnError* to 0 in config.ini only after the cluster has been started, then restart the data nodes normally—that is, without the *--initial* option. See the description of the *StopOnError* parameter for a detailed explanation of this issue. (Bug #24945638)

Use of this option prevents the *StartPartialTimeout* and *StartPartitionedTimeout* configuration parameters from having any effect.

**Important**

This option does *not* affect backup files that have already been created by the affected node.

Prior to NDB 8.0.21, the *--initial* option also did not affect any Disk Data files. In NDB 8.0.21 and later, when used to perform an initial restart of the cluster, the option causes the removal of all data files associated with Disk Data tablespaces and undo log files associated with log file groups that existed previously on this data node (see Section 7.10, “NDB Cluster Disk Data Tables”).

This option also has no effect on recovery of data by a data node that is just starting (or restarting) from data nodes that are already running (unless they also were started with *--initial*, as part of an initial restart). This recovery of data occurs automatically, and requires no user intervention in an NDB Cluster that is running normally.

It is permissible to use this option when starting the cluster for the very first time (that is, before any data node files have been created); however, it is *not* necessary to do so.

- *--initial-start*

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><em>--initial-start</em></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
This option is used when performing a partial initial start of the cluster. Each node should be started with this option, as well as --nowait-nodes.

Suppose that you have a 4-node cluster whose data nodes have the IDs 2, 3, 4, and 5, and you wish to perform a partial initial start using only nodes 2, 4, and 5—that is, omitting node 3:

```
shell> ndbd --ndb-nodeid=2 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=4 --nowait-nodes=3 --initial-start
shell> ndbd --ndb-nodeid=5 --nowait-nodes=3 --initial-start
```

When using this option, you must also specify the node ID for the data node being started with the --ndb-nodeid option.

---

**Important**

Do not confuse this option with the --nowait-nodes option for ndb_mgmd, which can be used to enable a cluster configured with multiple management servers to be started without all management servers being online.

---

- **--install[=name]**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--install[=name]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Platform Specific</td>
<td>Windows</td>
</tr>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>ndbd</td>
</tr>
</tbody>
</table>

Causes ndbd to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to ndbd. Although it is preferable to specify other ndbd program options in a my.ini or my.cnf configuration file, it is possible to use together with --install. However, in such cases, the --install option must be specified first, before any other options are given, for the Windows service installation to succeed.

It is generally not advisable to use this option together with the --initial option, since this causes the data node file system to be wiped and rebuilt every time the service is stopped and started. Extreme care should also be taken if you intend to use any of the other ndbd options that affect the starting of data nodes—including --initial-start, --nostart, and --nowait-nodes—together with --install, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The --install option has no effect on non-Windows platforms.

- **--logbuffer-size=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--logbuffer-size=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>32768</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>2048</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Sets the size of the data node log buffer. When debugging with high amounts of extra logging, it is possible for the log buffer to run out of space if there are too many log messages, in which case some log messages can be lost. This should not occur during normal operations.
**--nodaemon**

Command-Line Format: `--nodaemon`

Type: Boolean

Default Value: FALSE

Prevents `ndbd` or `ndbmtd` from executing as a daemon process. This option overrides the `--daemon` option. This is useful for redirecting output to the screen when debugging the binary.

The default behavior for `ndbd` and `ndbmtd` on Windows is to run in the foreground, making this option unnecessary on Windows platforms, where it has no effect.

**--nostart, -n**

Command-Line Format: `--nostart`

Type: Boolean

Default Value: FALSE

Instructs `ndbd` not to start automatically. When this option is used, `ndbd` connects to the management server, obtains configuration data from it, and initializes communication objects. However, it does not actually start the execution engine until specifically requested to do so by the management server. This can be accomplished by issuing the proper `START` command in the management client (see Section 7.1, “Commands in the NDB Cluster Management Client”).

**--nowait-nodes=node_id_1[, node_id_2[, ...]]**

Command-Line Format: `--nowait-nodes=list`

Type: String

Default Value

This option takes a list of data nodes for which the cluster does not wait, prior to starting.

This can be used to start the cluster in a partitioned state. For example, to start the cluster with only half of the data nodes (nodes 2, 3, 4, and 5) running in a 4-node cluster, you can start each `ndbd` process with `--nowait-nodes=3,5`. In this case, the cluster starts as soon as nodes 2 and 4 connect, and does not wait `StartPartitionedTimeout` milliseconds for nodes 3 and 5 to connect as it would otherwise.

If you wanted to start up the same cluster as in the previous example without one `ndbd` (say, for example, that the host machine for node 3 has suffered a hardware failure) then start nodes 2, 4, and 5 with `--nowait-nodes=3`. Then the cluster starts as soon as nodes 2, 4, and 5 connect, and does not wait for node 3 to start.

**--remove[=name]**

Command-Line Format: `--remove[=name]`

Platform Specific: Windows

Type: String

Default Value: `ndbd`

Causes an `ndbd` process that was previously installed as a Windows service to be removed. Optionally, you can specify a name for the service to be uninstalled; if not set, the service name defaults to `ndbd`.

The `--remove` option has no effect on non-Windows platforms.
--verbose, -v

Causes extra debug output to be written to the node log.

You can also use NODELOG DEBUG ON and NODELOG DEBUG OFF to enable and disable this extra logging while the data node is running.

ndbd generates a set of log files which are placed in the directory specified by DataDir in the config.ini configuration file.

These log files are listed below. node_id is and represents the node's unique identifier. For example, ndb_2_error.log is the error log generated by the data node whose node ID is 2.

- ndb_node_id_error.log is a file containing records of all crashes which the referenced ndbd process has encountered. Each record in this file contains a brief error string and a reference to a trace file for this crash. A typical entry in this file might appear as shown here:

```
Date/Time: Saturday 30 July 2004 - 00:20:01
Type of error: error
Message: Internal program error (failed ndbrequire)
Fault ID: 2341
Problem data: DbtupFixAlloc.cpp
Object of reference: DBTUP (Line: 173)
ProgramName: NDB Kernel
ProcessID: 14909
TraceFile: ndb_2_trace.log.2
***EOM***
```

Listings of possible ndbd exit codes and messages generated when a data node process shuts down prematurely can be found in Data Node Error Messages.

Important

_The last entry in the error log file is not necessarily the newest one_ (nor is it likely to be). Entries in the error log are _not_ listed in chronological order; rather, they correspond to the order of the trace files as determined in the ndb_node_id_trace.log.next file (see below). Error log entries are thus overwritten in a cyclical and not sequential fashion.

- ndb_node_id_trace.log.trace_id is a trace file describing exactly what happened just before the error occurred. This information is useful for analysis by the NDB Cluster development team.

It is possible to configure the number of these trace files that are created before old files are overwritten. trace_id is a number which is incremented for each successive trace file.

- ndb_node_id_trace.log.next is the file that keeps track of the next trace file number to be assigned.

- ndb_node_id_out.log is a file containing any data output by the ndbd process. This file is created only if ndbd is started as a daemon, which is the default behavior.

- ndb_node_id.pid is a file containing the process ID of the ndbd process when started as a daemon. It also functions as a lock file to avoid the starting of nodes with the same identifier.

- ndb_node_id_signal.log is a file used only in debug versions of ndbd, where it is possible to trace all incoming, outgoing, and internal messages with their data in the ndbd process.

It is recommended not to use a directory mounted through NFS because in some environments this can cause problems whereby the lock on the .pid file remains in effect even after the process has terminated.
To start `ndbd`, it may also be necessary to specify the host name of the management server and the port on which it is listening. Optionally, one may also specify the node ID that the process is to use.

```
shell> ndbd --connect-string="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

See Section 5.3.3, “NDB Cluster Connection Strings”, for additional information about this issue. Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, describes other command-line options which can be used with `ndbd`. For information about data node configuration parameters, see Section 5.3.6, “Defining NDB Cluster Data Nodes”.

When `ndbd` starts, it actually initiates two processes. The first of these is called the “angel process”; its only job is to discover when the execution process has been completed, and then to restart the `ndbd` process if it is configured to do so. Thus, if you attempt to kill `ndbd` using the Unix `kill` command, it is necessary to kill both processes, beginning with the angel process. The preferred method of terminating an `ndbd` process is to use the management client and stop the process from there.

The execution process uses one thread for reading, writing, and scanning data, as well as all other activities. This thread is implemented asynchronously so that it can easily handle thousands of concurrent actions. In addition, a watch-dog thread supervises the execution thread to make sure that it does not hang in an endless loop. A pool of threads handles file I/O, with each thread able to handle one open file. Threads can also be used for transporter connections by the transporters in the `ndbd` process. In a multi-processor system performing a large number of operations (including updates), the `ndbd` process can consume up to 2 CPUs if permitted to do so.

For a machine with many CPUs it is possible to use several `ndbd` processes which belong to different node groups; however, such a configuration is still considered experimental and is not supported for MySQL 8.0 in a production setting. See Section 3.7, “Known Limitations of NDB Cluster”.

### 6.2 ndbinfo_select_all — Select From ndbinfo Tables

`ndbinfo_select_all` is a client program that selects all rows and columns from one or more tables in the `ndbinfo` database.

Not all `ndbinfo` tables available in the `mysql` client can be read by this program. In addition, `ndbinfo_select_all` can show information about some tables internal to `ndbinfo` which cannot be accessed using SQL, including the `tables` and `columns` metadata tables.

To select from one or more `ndbinfo` tables using `ndbinfo_select_all`, it is necessary to supply the names of the tables when invoking the program as shown here:

```
shell> ndbinfo_select_all table_name1 [table_name2] [...]
```

For example:

```
shell> ndbinfo_select_all logbuffers logspaces

```

<table>
<thead>
<tr>
<th>node_id</th>
<th>log_type</th>
<th>log_id</th>
<th>log_part</th>
<th>total</th>
<th>used</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33554432</td>
<td>262144</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33554432</td>
<td>262144</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33554432</td>
<td>262144</td>
<td>0</td>
</tr>
<tr>
<td>8</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>33554432</td>
<td>262144</td>
<td>0</td>
</tr>
</tbody>
</table>

```

<table>
<thead>
<tr>
<th>node_id</th>
<th>log_type</th>
<th>log_id</th>
<th>log_part</th>
<th>total</th>
<th>used</th>
<th>high</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>5</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>2</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>7</td>
<td>0</td>
<td>0</td>
<td>3</td>
<td>268435456</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>
```
The following table includes options that are specific to `ndbinfo_select_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndbinfo_select_all`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.2 Command-line options used with the program `ndbinfo_select_all`

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--delay=#</code></td>
<td>Set delay in seconds between loops</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--loops=#</code>, <code>-l</code></td>
<td>Set number of times to perform select</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--database=db_name</code>, <code>-d</code></td>
<td>Name of database where table is located</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--parallelism=#</code>, <code>-p</code></td>
<td>Set degree of parallelism</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

• **--delay=seconds**

Command-Line Format: `--delay=#`

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>5</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>

This option sets the number of seconds to wait between executing loops. Has no effect if `--loops` is set to 0 or 1.

• **--loops=number, -l number**

Command-Line Format: `--loops=#`

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>

This option sets the number of times to execute the select. Use `--delay` to set the time between loops.

### 6.3 ndbmtd — The NDB Cluster Data Node Daemon (Multi-Threaded)

`ndbmtd` is a multithreaded version of `ndbd`, the process that is used to handle all the data in tables using the `NDBCLUSTER` storage engine. `ndbmtd` is intended for use on host computers having multiple CPU cores. Except where otherwise noted, `ndbmtd` functions in the same way as `ndbd`; therefore, in
this section, we concentrate on the ways in which ndbmtd differs from ndbd, and you should consult Section 6.1, "ndbd — The NDB Cluster Data Node Daemon", for additional information about running NDB Cluster data nodes that apply to both the single-threaded and multithreaded versions of the data node process.

Command-line options and configuration parameters used with ndbd also apply to ndbmtd. For more information about these options and parameters, see Section 6.1, "ndbd — The NDB Cluster Data Node Daemon", and Section 5.3.6, "Defining NDB Cluster Data Nodes", respectively.

ndbmtd is also file system-compatible with ndbd. In other words, a data node running ndbd can be stopped, the binary replaced with ndbmtd, and then restarted without any loss of data. (However, when doing this, you must make sure that MaxNoOfExecutionThreads is set to an appropriate value before restarting the node if you wish for ndbmtd to run in multithreaded fashion.) Similarly, an ndbmtd binary can be replaced with ndbd simply by stopping the node and then starting ndbd in place of the multithreaded binary. It is not necessary when switching between the two to start the data node binary using --initial.

Using ndbmtd differs from using ndbd in two key respects:

1. Because ndbmtd runs by default in single-threaded mode (that is, it behaves like ndbd), you must configure it to use multiple threads. This can be done by setting an appropriate value in the config.ini file for the MaxNoOfExecutionThreads configuration parameter or the ThreadConfig configuration parameter. Using MaxNoOfExecutionThreads is simpler, but ThreadConfig offers more flexibility. For more information about these configuration parameters and their use, see Multi-Threading Configuration Parameters (ndbmtd).

2. Trace files are generated by critical errors in ndbmtd processes in a somewhat different fashion from how these are generated by ndbd failures. These differences are discussed in more detail in the next few paragraphs.

Like ndbd, ndbmtd generates a set of log files which are placed in the directory specified by DataDir in the config.ini configuration file. Except for trace files, these are generated in the same way and have the same names as those generated by ndbd.

In the event of a critical error, ndbmtd generates trace files describing what happened just prior to the error occurrence. These files, which can be found in the data node's DataDir, are useful for analysis of problems by the NDB Cluster Development and Support teams. One trace file is generated for each ndbmtd thread. The names of these files have the following pattern:

`ndb_node_id_trace.log.trace_id_thread_id`

In this pattern, node_id stands for the data node's unique node ID in the cluster, trace_id is a trace sequence number, and thread_id is the thread ID. For example, in the event of the failure of an ndbmtd process running as an NDB Cluster data node having the node ID 3 and with MaxNoOfExecutionThreads equal to 4, four trace files are generated in the data node's data directory. If the is the first time this node has failed, then these files are named ndb_3_trace.log.1_t1, ndb_3_trace.log.1_t2, ndb_3_trace.log.1_t3, and ndb_3_trace.log.1_t4. Internally, these trace files follow the same format as ndbd trace files.

The ndbd exit codes and messages that are generated when a data node process shuts down prematurely are also used by ndbmtd. See Data Node Error Messages, for a listing of these.

Note

It is possible to use ndbd and ndbmtd concurrently on different data nodes in the same NDB Cluster. However, such configurations have not been tested extensively; thus, we cannot recommend doing so in a production setting at this time.

6.4 ndb_mgmd — The NDB Cluster Management Server Daemon
The management server is the process that reads the cluster configuration file and distributes this information to all nodes in the cluster that request it. It also maintains a log of cluster activities. Management clients can connect to the management server and check the cluster's status.

The following table includes options that are specific to the NDB Cluster management server program `ndb_mgmd`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_mgmd`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

### Table 6.3 Command-line options used with the program `ndb_mgmd`

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--bind-address=host</code></td>
<td>Local bind address</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--cluster-config-suffix=name</code></td>
<td>Override defaults group suffix when reading cluster_config sections in my.cnf file; used in testing</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td>`--config-cache=[TRUE</td>
<td>FALSE]`</td>
<td>Enable management server configuration cache; true by default</td>
</tr>
<tr>
<td><code>--config-file=file, -f</code></td>
<td>Specify cluster configuration file; also specify <code>--reload</code> or <code>--initial</code> to override configuration cache if present</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--configdir=directory, --config-dir=directory</code></td>
<td>Specify cluster management server configuration cache directory</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--daemon, -d</code></td>
<td>Run ndb_mgmd in daemon mode (default)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--initial</code></td>
<td>Causes management server to reload configuration data from configuration file, bypassing configuration cache</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--install=[name]</code></td>
<td>Used to install management server process as Windows service; does not apply on other platforms</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--interactive</code></td>
<td>Run ndb_mgmd in interactive mode (not officially supported in production; for testing purposes only)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--log-name=name</code></td>
<td>Name to use when writing cluster log messages applying to this node</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--mycnf</code></td>
<td>Read cluster configuration data from my.cnf file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--no-nodeid-checks</code></td>
<td>Do not provide any node ID checks</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--nodaemon</code></td>
<td>Do not run ndb_mgmd as a daemon</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--nowait-nodes=list</code></td>
<td>Do not wait for management nodes specified when starting this</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### ndb_mgmd — The NDB Cluster Management Server Daemon

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--print-full-config,</td>
<td>management server; requires --ndb-nodeid option</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-p</td>
<td>Print full configuration and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--reload</td>
<td>Causes management server to compare configuration file with configuration cache</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--remove[=name]</td>
<td>Used to remove management server process that was previously installed as Windows service, optionally specifying name of service to be removed; does not apply on other platforms</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Write additional information to log</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

- **--bind-address=host**

  **Command-Line Format**: `--bind-address=host`  
  **Type**: String  
  **Default Value**: [none]  
  Causes the management server to bind to a specific network interface (host name or IP address). This option has no default value.

- **cluster-config-suffix**

  **Command-Line Format**: `--cluster-config-suffix=name`  
  **Introduced**: 8.0.24-ndb-8.0.24  
  **Type**: String  
  **Default Value**: [none]  
  Override defaults group suffix when reading cluster configuration sections in `my.cnf`; used in testing.

- **--config-cache**

  **Command-Line Format**: `--config-cache=[TRUE|FALSE]`  
  **Type**: Boolean  
  **Default Value**: TRUE  
  This option, whose default value is 1 (or TRUE, or ON), can be used to disable the management server's configuration cache, so that it reads its configuration from `config.ini` every time it starts (see Section 5.3, “NDB Cluster Configuration Files”). You can do this by starting the `ndb_mgmd` process with any one of the following options:

  - **--config-cache=0**
  - **--config-cache=FALSE**
  - **--config-cache=OFF**
• **--skip-config-cache**

Using one of the options just listed is effective only if the management server has no stored configuration at the time it is started. If the management server finds any configuration cache files, then the `--config-cache` option or the `--skip-config-cache` option is ignored. Therefore, to disable configuration caching, the option should be used the first time that the management server is started. Otherwise—that is, if you wish to disable configuration caching for a management server that has already created a configuration cache—you must stop the management server, delete any existing configuration cache files manually, then restart the management server with `--skip-config-cache` (or with `--config-cache` set equal to 0, OFF, or FALSE).

Configuration cache files are normally created in a directory named `mysql-cluster` under the installation directory (unless this location has been overridden using the `--configdir` option). Each time the management server updates its configuration data, it writes a new cache file. The files are named sequentially in order of creation using the following format:

```
ndb_node-id_config.bin.seq-number
```

`node-id` is the management server's node ID; `seq-number` is a sequence number, beginning with 1. For example, if the management server's node ID is 5, then the first three configuration cache files would, when they are created, be named `ndb_5_config.bin.1`, `ndb_5_config.bin.2`, and `ndb_5_config.bin.3`.

If your intent is to purge or reload the configuration cache without actually disabling caching, you should start `ndb_mgmd` with one of the options `--reload` or `--initial` instead of `--skip-config-cache`.

To re-enable the configuration cache, simply restart the management server, but without the `--config-cache` or `--skip-config-cache` option that was used previously to disable the configuration cache.

`ndb_mgmd` does not check for the configuration directory (`--configdir`) or attempts to create one when `--skip-config-cache` is used. (Bug #13428853)

• **--config-file=filename, -f filename**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--config-file=file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Instructs the management server as to which file it should use for its configuration file. By default, the management server looks for a file named `config.ini` in the same directory as the `ndb_mgmd` executable; otherwise the file name and location must be specified explicitly.

This option has no default value, and is ignored unless the management server is forced to read the configuration file, either because `ndb_mgmd` was started with the `--reload` or `--initial` option, or because the management server could not find any configuration cache. This option is also read if `ndb_mgmd` was started with `--config-cache=OFF`. See Section 5.3, "NDB Cluster Configuration Files", for more information.

• **--configdir=dir_name**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--configdir=directory</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>--config-dir=directory</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
</tbody>
</table>

`ndb_mgmd` does not check for the configuration directory (`--configdir`) or attempts to create one when `--skip-config-cache` is used. (Bug #13428853)
### ndb_mgmd — The NDB Cluster Management Server Daemon

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--config-dir</td>
<td>Specifies the cluster management server's configuration cache directory. --config-dir is an alias for this option.</td>
</tr>
<tr>
<td>--daemon, -d</td>
<td>Instructs ndb_mgmd to start as a daemon process. This is the default behavior. This option has no effect when running ndb_mgmd on Windows platforms.</td>
</tr>
<tr>
<td>--initial</td>
<td>Configuration data is cached internally, rather than being read from the cluster global configuration file each time the management server is started (see Section 5.3, “NDB Cluster Configuration Files”). Using the --initial option overrides this behavior, by forcing the management server to delete any existing cache files, and then to re-read the configuration data from the cluster configuration file and to build a new cache. This differs in two ways from the --reload option. First, --reload forces the server to check the configuration file against the cache and reload its data only if the contents of the file are different from the cache. Second, --reload does not delete any existing cache files. If ndb_mgmd is invoked with --initial but cannot find a global configuration file, the management server cannot start. When a management server starts, it checks for another management server in the same NDB Cluster and tries to use the other management server’s configuration data. This behavior has implications when performing a rolling restart of an NDB Cluster with multiple management nodes. See Section 7.5, “Performing a Rolling Restart of an NDB Cluster”, for more information. When used together with the --config-file option, the cache is cleared only if the configuration file is actually found.</td>
</tr>
<tr>
<td>--install[=name]</td>
<td>Causes ndb_mgmd to be installed as a Windows service. Optionally, you can specify a name for the service; if not set, the service name defaults to ndb_mgmd. Although it is preferable to specify other ndb_mgmd program options in a my.ini or my.cnf configuration file, it is possible to use them together with --install. However, in such cases, the --install option must be specified first, before any other options are given, for the Windows service installation to succeed. It is generally not advisable to use this option together with the --initial option, since this causes the configuration cache to be wiped and rebuilt every time the service is stopped and started. Care</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Default Value</strong></th>
<th><strong>$INSTALLDIR/mysql-cluster</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the cluster management server's configuration cache directory. --config-dir is an alias for this option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Default Value</strong></th>
<th><strong>$INSTALLDIR/mysql-cluster</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the cluster management server's configuration cache directory. --config-dir is an alias for this option.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th><strong>Default Value</strong></th>
<th><strong>$INSTALLDIR/mysql-cluster</strong></th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Description</strong></td>
<td>Specifies the cluster management server's configuration cache directory. --config-dir is an alias for this option.</td>
</tr>
</tbody>
</table>
should also be taken if you intend to use any other `ndb_mgmd` options that affect the starting of the management server, and you should make absolutely certain you fully understand and allow for any possible consequences of doing so.

The `--install` option has no effect on non-Windows platforms.

• **--interactive**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--interactive</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Starts `ndb_mgmd` in interactive mode; that is, an `ndb_mgm` client session is started as soon as the management server is running. This option does not start any other NDB Cluster nodes.

• **--log-name=name**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--log-name=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>MgmtSrvr</td>
</tr>
</tbody>
</table>

Provides a name to be used for this node in the cluster log.

• **--mycnf**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--mycnf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Read configuration data from the `my.cnf` file.

• **--no-nodeid-checks**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-nodeid-checks</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Do not perform any checks of node IDs.

• **--nodaemon**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--nodaemon</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Instructs `ndb_mgmd` not to start as a daemon process.

The default behavior for `ndb_mgmd` on Windows is to run in the foreground, making this option unnecessary on Windows platforms.

• **--nowait-nodes**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--nowait-nodes=list</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>
When starting an NDB Cluster is configured with two management nodes, each management server normally checks to see whether the other ndb_mgmd is also operational and whether the other management server's configuration is identical to its own. However, it is sometimes desirable to start the cluster with only one management node (and perhaps to allow the other ndb_mgmd to be started later). This option causes the management node to bypass any checks for any other management nodes whose node IDs are passed to this option, permitting the cluster to start as though configured to use only the management node that was started.

For purposes of illustration, consider the following portion of a config.ini file (where we have omitted most of the configuration parameters that are not relevant to this example):

```
[ndbd]
NodeId = 1
HostName = 198.51.100.101
[ndbd]
NodeId = 2
HostName = 198.51.100.102
[ndbd]
NodeId = 3
HostName = 198.51.100.103
[ndbd]
NodeId = 4
HostName = 198.51.100.104
[ndb_mgmd]
NodeId = 10
HostName = 198.51.100.150
[ndb_mgmd]
NodeId = 11
HostName = 198.51.100.151
[api]
NodeId = 20
HostName = 198.51.100.200
[api]
NodeId = 21
HostName = 198.51.100.201
```

Assume that you wish to start this cluster using only the management server having node ID 10 and running on the host having the IP address 198.51.100.150. (Suppose, for example, that the host computer on which you intend to the other management server is temporarily unavailable due to a hardware failure, and you are waiting for it to be repaired.) To start the cluster in this way, use a command line on the machine at 198.51.100.150 to enter the following command:

```
shell> ndb_mgmd --ndb-nodeid=10 --nowait-nodes=11
```

As shown in the preceding example, when using --nowait-nodes, you must also use the --ndb-nodeid option to specify the node ID of this ndb_mgmd process.

You can then start each of the cluster's data nodes in the usual way. If you wish to start and use the second management server in addition to the first management server at a later time without restarting the data nodes, you must start each data node with a connection string that references both management servers, like this:

```
shell> ndbd -c 198.51.100.150,198.51.100.151
```

The same is true with regard to the connection string used with any mysqld processes that you wish to start as NDB Cluster SQL nodes connected to this cluster. See Section 5.3.3, “NDB Cluster Connection Strings”, for more information.

When used with ndb_mgmd, this option affects the behavior of the management node with regard to other management nodes only. Do not confuse it with the --nowait-nodes option used with ndbd.
ndb_mgmd — The NDB Cluster Management Server Daemon

or ndbmtd to permit a cluster to start with fewer than its full complement of data nodes; when used with data nodes, this option affects their behavior only with regard to other data nodes.

Multiple management node IDs may be passed to this option as a comma-separated list. Each node ID must be no less than 1 and no greater than 255. In practice, it is quite rare to use more than two management servers for the same NDB Cluster (or to have any need for doing so); in most cases you need to pass to this option only the single node ID for the one management server that you do not wish to use when starting the cluster.

**Note**

When you later start the “missing” management server, its configuration must match that of the management server that is already in use by the cluster. Otherwise, it fails the configuration check performed by the existing management server, and does not start.

- **--print-full-config, -P**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--print-full-config</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  Shows extended information regarding the configuration of the cluster. With this option on the command line the ndb_mgmd process prints information about the cluster setup including an extensive list of the cluster configuration sections as well as parameters and their values. Normally used together with the **--config-file (-f)** option.

- **--reload**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--reload</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  NDB Cluster configuration data is stored internally rather than being read from the cluster global configuration file each time the management server is started (see Section 5.3, “NDB Cluster Configuration Files”). Using this option forces the management server to check its internal data store against the cluster configuration file and to reload the configuration if it finds that the configuration file does not match the cache. Existing configuration cache files are preserved, but not used.

  This differs in two ways from the **--initial** option. First, --initial causes all cache files to be deleted. Second, --initial forces the management server to re-read the global configuration file and construct a new cache.

  If the management server cannot find a global configuration file, then the **--reload** option is ignored.

  When --reload is used, the management server must be able to communicate with data nodes and any other management servers in the cluster before it attempts to read the global configuration file; otherwise, the management server fails to start. This can happen due to changes in the networking environment, such as new IP addresses for nodes or an altered firewall configuration. In such cases, you must use **--initial** instead to force the exiting cached configuration to be discarded and reloaded from the file. See Section 7.5, “Performing a Rolling Restart of an NDB Cluster”, for additional information.

- **--remove{=name}**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--remove{=name}</th>
</tr>
</thead>
</table>
### ndb_mgm — The NDB Cluster Management Client

The `ndb_mgm` management client process is actually not needed to run the cluster. Its value lies in providing a set of commands for checking the cluster's status, starting backups, and performing other administrative functions. The management client accesses the management server using a C API. Advanced users can also employ this API for programming dedicated management processes to perform tasks similar to those performed by `ndb_mgm`.

To start the management client, it is necessary to supply the host name and port number of the management server:

```
shell> ndb_mgm [host_name [port_num]]
```
For example:

```
shell> ndb_mgm ndb_mgmd.mysql.com 1186
```

The default host name and port number are `localhost` and 1186, respectively.

The following table includes options that are specific to the NDB Cluster management client program `ndb_mgm`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_mgm`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.4 Command-line options used with the program `ndb_mgm`

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--backup-password-from-stdin[]=value</code></td>
<td>Get decryption password in a secure fashion from STDIN; use together with <code>--execute</code> and <code>START BACKUP</code> command</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td><code>--connect-retries=##</code></td>
<td>Set number of times to retry connection before giving up; 0 means 1 attempt only (and no retries)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>`--encrypt-backup[]=TRUE</td>
<td>FALSE`</td>
<td>Cause <code>START BACKUP</code> to encrypt whenever making a backup, prompting for password if not supplied by user</td>
</tr>
<tr>
<td><code>--execute=name,-e</code></td>
<td>Execute command and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--try-reconnect=##,-t</code></td>
<td>Set number of times to retry connection before giving up; synonym for <code>--connect-retries</code></td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

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

Command-Line Format: `--backup-password-from-stdin[]=value`

<table>
<thead>
<tr>
<th>Introduced</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>8.0.24-ndb-8.0.24</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

This option enables input of the backup password from the system shell (`stdin`) when using `--execute "START BACKUP"` or similar to create a backup. Use of this option requires use of `--execute` as well.

• `--connect-retries=##`

Command-Line Format: `--connect-retries=##`

<table>
<thead>
<tr>
<th>Type</th>
<th>Default Value</th>
<th>Minimum Value</th>
<th>Maximum Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Numeric</td>
<td>3</td>
<td>0</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using `--connect-retry-delay`. 
This option is synonymous with the --try-reconnect option, which is now deprecated.

The default for this option this option differs from its default when used with other NDB programs. See Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”, for more information.

• --encrypt-backup

**Command-Line Format**

`--encrypt-backup[=TRUE|FALSE]`

**Introduced**

8.0.24-ndb-8.0.24

**Type**

Boolean

**Default Value**

TRUE

When used, this option causes all backups to be encrypted. To make this happen whenever ndb_mgm is run, put the option in the [ndb_mgm] section of the my.cnf file.

• --execute=command, -e command

**Command-Line Format**

`--execute=name`

This option can be used to send a command to the NDB Cluster management client from the system shell. For example, either of the following is equivalent to executing SHOW in the management client:

```
shell> ndb_mgm -e "SHOW"
shell> ndb_mgm --execute="SHOW"
```

This is analogous to how the --execute or -e option works with the mysql command-line client. See Using Options on the Command Line.

**Note**

If the management client command to be passed using this option contains any space characters, then the command must be enclosed in quotation marks. Either single or double quotation marks may be used. If the management client command contains no space characters, the quotation marks are optional.

• --try-reconnect=number

**Command-Line Format**

`--try-reconnect=#`

**Deprecated**

Yes

**Type**

Numeric

**Type**

Integer

**Default Value**

12

**Default Value**

3

**Minimum Value**

0

**Maximum Value**

4294967295

If the connection to the management server is broken, the node tries to reconnect to it every 5 seconds until it succeeds. By using this option, it is possible to limit the number of attempts to number before giving up and reporting an error instead.

This option is deprecated and subject to removal in a future release. Use --connect-retries, instead.
6.6 ndb_blob_tool — Check and Repair BLOB and TEXT columns of NDB Cluster Tables

This tool can be used to check for and remove orphaned BLOB column parts from NDB tables, as well as to generate a file listing any orphaned parts. It is sometimes useful in diagnosing and repairing corrupted or damaged NDB tables containing BLOB or TEXT columns.

The basic syntax for ndb_blob_tool is shown here:

```
ndb_blob_tool [options] table [column, ...]
```

Unless you use the --help option, you must specify an action to be performed by including one or more of the options --check-orphans, --delete-orphans, or --dump-file. These options cause ndb_blob_tool to check for orphaned BLOB parts, remove any orphaned BLOB parts, and generate a dump file listing orphaned BLOB parts, respectively, and are described in more detail later in this section.

You must also specify the name of a table when invoking ndb_blob_tool. In addition, you can optionally follow the table name with the (comma-separated) names of one or more BLOB or TEXT columns from that table. If no columns are listed, the tool works on all of the table's BLOB and TEXT columns. If you need to specify a database, use the --database (-d) option.

The --verbose option provides additional information in the output about the tool's progress.

The following table includes options that are specific to ndb_blob_tool. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_blob_tool), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.5 Command-line options used with the program ndb_blob_tool

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--add-missing</td>
<td>Write dummy blob parts to take place of those which are missing</td>
<td>ADDED: NDB 8.0.20</td>
</tr>
<tr>
<td>--check-missing</td>
<td>Check for blobs having inline parts but missing one or more parts from parts table</td>
<td>ADDED: NDB 8.0.20</td>
</tr>
<tr>
<td>--check-orphans</td>
<td>Check for blob parts having no corresponding inline parts</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--database=db_name, -d</td>
<td>Database to find the table in</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--delete-orphans</td>
<td>Delete blob parts having no corresponding inline parts</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--dump-file=file</td>
<td>Write orphan keys to specified file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Verbose output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

• --add-missing
**ndb_blob_tool** — Check and Repair BLOB and TEXT columns of NDB Cluster Tables

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--add-missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.20-ndb-8.0.20</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

For each inline part in NDB Cluster tables which has no corresponding BLOB part, write a dummy BLOB part of the required length, consisting of spaces.

- **--check-missing**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--check-missing</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.20-ndb-8.0.20</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Check for inline parts in NDB Cluster tables which have no corresponding BLOB parts.

- **--check-orphans**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--check-orphans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Check for BLOB parts in NDB Cluster tables which have no corresponding inline parts.

- **--database=db_name, -d**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--database=db_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Specify the database to find the table in.

- **--delete-orphans**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--delete-orphans</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Remove BLOB parts from NDB Cluster tables which have no corresponding inline parts.

- **--dump-file=file**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--dump-file=file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Writes a list of orphaned BLOB column parts to file. The information written to the file includes the table key and BLOB part number for each orphaned BLOB part.

- **--verbose**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--verbose</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Example

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Provide extra information in the tool's output regarding its progress.

Example

First we create an NDB table in the test database, using the CREATE TABLE statement shown here:

```sql
USE test;
CREATE TABLE btest (
    c0 BIGINT UNSIGNED NOT NULL AUTO_INCREMENT PRIMARY KEY,
    c1 TEXT,
    c2 BLOB
) ENGINE=NDB;
```

Then we insert a few rows into this table, using a series of statements similar to this one:

```sql
INSERT INTO btest VALUES (NULL, 'x', REPEAT('x', 1000));
```

When run with `--check-orphans` against this table, `ndb_blob_tool` generates the following output:

```
shell> ndb_blob_tool --check-orphans --verbose -d test btest
connected
processing 2 blobs
processing blob #0 c1 NDB$BLOB_19_1
NDB$BLOB_19_1: nextResult: res=1
total parts: 0
orphan parts: 0
processing blob #1 c2 NDB$BLOB_19_2
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=0
NDB$BLOB_19_2: nextResult: res=1
total parts: 10
orphan parts: 0
disconnected
NDBT_ProgramExit: 0 - OK
```

The tool reports that there are no NDB BLOB column parts associated with column c1, even though c1 is a TEXT column. This is due to the fact that, in an NDB table, only the first 256 bytes of a BLOB or TEXT column value are stored inline, and only the excess, if any, is stored separately; thus, if there are no values using more than 256 bytes in a given column of one of these types, no BLOB column parts are created by NDB for this column. See Data Type Storage Requirements, for more information.

6.7 `ndb_config` — Extract NDB Cluster Configuration Information

This tool extracts current configuration information for data nodes, SQL nodes, and API nodes from one of a number of sources: an NDB Cluster management node, or its config.ini or my.cnf file. By default, the management node is the source for the configuration data; to override the default, execute `ndb_config` with the `--config-file` or `--mycnf` option. It is also possible to use a data node as the source by specifying its node ID with `--config_from_node=node_id`.

`ndb_config` can also provide an offline dump of all configuration parameters which can be used, along with their default, maximum, and minimum values and other information. The dump can be produced in either text or XML format; for more information, see the discussion of the `--configinfo` and `--xml` options later in this section.)
You can filter the results by section (DB, SYSTEM, or CONNECTIONS) using one of the options --nodes, --system, or --connections.

The following table includes options that are specific to `ndb_config`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_config`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

### Table 6.6 Command-line options used with the program `ndb_config`

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--cluster-config-suffix=name</td>
<td>Override defaults group suffix when reading cluster_config sections in my.cnf file; used in testing</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td>--config-file=file_name</td>
<td>Set the path to config.ini file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--config-from-node=#</td>
<td>Obtain configuration data from the node having this ID (must be a data node)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--configinfo</td>
<td>Dumps information about all NDB configuration parameters in text format with default, maximum, and minimum values. Use with --xml to obtain XML output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--connections</td>
<td>Print connections information ([tcp], [tcp default], [sci], [sci default], [shm], or [shm default] sections of cluster configuration file) only. Cannot be used with --system or --nodes</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--diff-default</td>
<td>Print only configuration parameters that have non-default values</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--fields=string, -f</td>
<td>Field separator</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--host=name</td>
<td>Specify host</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--mycnf</td>
<td>Read configuration data from my.cnf file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nodeid</td>
<td>Get configuration of node with this ID</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nodes</td>
<td>Print node information ([ndbd] or [ndbd default] section of cluster configuration file) only. Cannot be used with --system or --connections</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-c</td>
<td>Short form for --ndb-connectstring</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--query=string, -q</td>
<td>One or more query options (attributes)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### ndb_config — Extract NDB Cluster Configuration Information

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--query-all, -a</code></td>
<td>Dumps all parameters and values to a single comma-delimited string</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--rows=string, -r</code></td>
<td>Row separator</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--system</code></td>
<td>Print SYSTEM section information only (see ndb_config --configinfo output). Cannot be used with --nodes or --connections</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--type=name</code></td>
<td>Specify node type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--configinfo --xml</code></td>
<td>Use --xml with --configinfo to obtain a dump of all NDB configuration parameters in XML format with default, maximum, and minimum values</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

*cluster-config-suffix*

**Command-Line Format**

`--cluster-config-suffix=name`

**Introduced**

8.0.24-ndb-8.0.24

**Type**

String

**Default Value**

[none]

Override defaults group suffix when reading cluster configuration sections in `my.cnf`; used in testing.

*--configinfo*

The `--configinfo` option causes `ndb_config` to dump a list of each NDB Cluster configuration parameter supported by the NDB Cluster distribution of which `ndb_config` is a part, including the following information:

- A brief description of each parameter's purpose, effects, and usage
- The section of the `config.ini` file where the parameter may be used
- The parameter's data type or unit of measurement
- Where applicable, the parameter's default, minimum, and maximum values
- NDB Cluster release version and build information

By default, this output is in text format. Part of this output is shown here:

```
shell> ndb_config --configinfo
****** SYSTEM ******
Name (String)
Name of system (NDB Cluster)
MANDATORY
PrimaryMGMNode (Non-negative Integer)
Node id of Primary ndb_mgmd(MGM) node
Default: 0 (Min: 0, Max: 4294967039)
ConfigGenerationNumber (Non-negative Integer)
Configuration generation number
```
ndb_config — Extract NDB Cluster Configuration Information

Use this option together with the --xml option to obtain output in XML format.

- **--config-file=path-to-file**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--config-file=file_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

  Gives the path to the management server’s configuration file (config.ini). This may be a relative or absolute path. If the management node resides on a different host from the one on which ndb_config is invoked, then an absolute path must be used.

- **--config_from_node=**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--config-from-node=###</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
<tr>
<td>Default Value</td>
<td>none</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>48</td>
</tr>
</tbody>
</table>

  Obtain the cluster’s configuration data from the data node that has this ID.

  If the node having this ID is not a data node, ndb_config fails with an error. (To obtain configuration data from the management node instead, simply omit this option.)

- **--connections**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--connections</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  Tells ndb_config to print CONNECTIONS information only—that is, information about parameters found in the [tcp], [tcp default], [shm], or [shm default] sections of the cluster configuration file (see Section 5.3.10, “NDB Cluster TCP/IP Connections”, and Section 5.3.12, “NDB Cluster Shared-Memory Connections”, for more information).

  This option is mutually exclusive with --nodes and --system; only one of these 3 options can be used.

- **--diff-default**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--diff-default</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  Print only configuration parameters that have non-default values.

- **--fields=delimiter, -f delimiter**
### ndb_config — Extract NDB Cluster Configuration Information

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--fields=string</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

Specifies a *delimiter* string used to separate the fields in the result. The default is `,` (the comma character).

**Note**

If the *delimiter* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- **--host=hostname**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--host=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

Specifies the host name of the node for which configuration information is to be obtained.

**Note**

While the hostname `localhost` usually resolves to the IP address `127.0.0.1`, this may not necessarily be true for all operating platforms and configurations. This means that it is possible, when `localhost` is used in `config.ini`, for `ndb_config --host=localhost` to fail if `ndb_config` is run on a different host where `localhost` resolves to a different address (for example, on some versions of SUSE Linux, this is `127.0.0.2`). In general, for best results, you should use numeric IP addresses for all NDB Cluster configuration values relating to hosts, or verify that all NDB Cluster hosts handle `localhost` in the same fashion.

- **--mycnf**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--mycnf</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Read configuration data from the `my.cnf` file.

- **--ndb-connectstring=connection_string,** `-c connection_string`  

| Command-Line Format          | --ndb-connectstring=connectstring  
<table>
<thead>
<tr>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>localhost:1186</td>
</tr>
</tbody>
</table>

Specifies the connection string to use in connecting to the management server. The format for the connection string is the same as described in Section 5.3.3, “NDB Cluster Connection Strings”, and defaults to `localhost:1186`.

- **--nodeid=node_id**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-nodeid=##</th>
</tr>
</thead>
</table>
Specify the node ID of the node for which configuration information is to be obtained.

- **--nodes**

  Command-Line Format: `--nodes`

  Type: Boolean

  Default Value: FALSE

  Tells `ndb_config` to print information relating only to parameters defined in an `[ndbd]` or `[ndbd default]` section of the cluster configuration file (see Section 5.3.6, “Defining NDB Cluster Data Nodes”).

  This option is mutually exclusive with `--connections` and `--system`; only one of these 3 options can be used.

- **--query=query-options, -q query-options**

  Command-Line Format: `--query=string`

  Type: String

  Default Value:

  This is a comma-delimited list of *query options*—that is, a list of one or more node attributes to be returned. These include `nodeid` (node ID), `type` (node type—that is, `ndbd`, `mysql`, or `ndb_mgm`), and any configuration parameters whose values are to be obtained.

  For example, `--query=nodeid,type,datamemory,datadir` returns the node ID, node type, DataMemory, and DataDir for each node.

  **Note**

  If a given parameter is not applicable to a certain type of node, than an empty string is returned for the corresponding value. See the examples later in this section for more information.

- **--query-all, -a**

  Command-Line Format: `--query-all`

  Type: String

  Default Value:

  Returns a comma-delimited list of all query options (node attributes; note that this list is a single string.

- **--rows=separator, -r separator**

  Command-Line Format: `--rows=string`

  Type: String
### ndb_config — Extract NDB Cluster Configuration Information

**Default Value**

Specifies a *separator* string used to separate the rows in the result. The default is a space character.

**Note**

If the *separator* contains spaces or escapes (such as `\n` for the linefeed character), then it must be quoted.

- **--system**

  **Command-Line Format**
  
  `--system`

  **Type**
  
  Boolean

  **Default Value**
  
  FALSE

  Tells *ndb_config* to print *SYSTEM* information only. This consists of system variables that cannot be changed at run time; thus, there is no corresponding section of the cluster configuration file for them. They can be seen (prefixed with `****** SYSTEM ******`) in the output of *ndb_config --configinfo*.

  This option is mutually exclusive with **--nodes** and **--connections**; only one of these 3 options can be used.

- **--type=node_type**

  **Command-Line Format**
  
  `--type=name`

  **Type**
  
  Enumeration

  **Default Value**
  
  [none]

  **Valid Values**
  
  ndbd
  
  mysqlld
  
  ndb_mgmd

  Filters results so that only configuration values applying to nodes of the specified *node_type* (ndbd, mysqlld, or ndb_mgmd) are returned.

- **--usage, --help, or -?**

  **Command-Line Format**
  
  `--help`

  `--usage`

  Causes *ndb_config* to print a list of available options, and then exit.

- **--version, -V**

  **Command-Line Format**
  
  `--version`

  Causes *ndb_config* to print a version information string, and then exit.

- **--configinfo --xml**

  **Command-Line Format**
  
  `--configinfo --xml`

  **Type**
  
  Boolean

  **Default Value**
  
  false
Cause `ndb_config --configinfo` to provide output as XML by adding this option. A portion of such output is shown in this example:

```
shell> ndb_config --configinfo --xml
<configvariables protocolversion="1" ndbversionstring="5.7.33-ndb-7.5.22"
    ndbversion="460032" ndbversionmajor="7" ndbversionminor="5"
    ndbversionbuild="0">
    <section name="SYSTEM">
        <param name="Name" comment="Name of system (NDB Cluster)" type="string"
            mandatory="true"/>
        <param name="PrimaryMGMNode" comment="Node id of Primary ndb_mgmd (MGM) node"
            type="unsigned" default="0" min="0" max="4294967039"/>
        <param name="ConfigGenerationNumber" comment="Configuration generation number"
            type="unsigned" default="0" min="0" max="4294967039"/>
    </section>
    <section name="MYSQLD" primarykeys="NodeId">
        <param name="wan" comment="Use WAN TCP setting as default" type="bool"
            default="false"/>
        <param name="HostName" comment="Name of computer for this node"
            type="string" default=""/>
        <param name="Id" comment="NodeId" type="unsigned" mandatory="true"
            min="1" max="255" deprecated="true"/>
        <param name="NodeId" comment="Number identifying application node (mysqld(API))"
            type="unsigned" mandatory="true" min="1" max="255"/>
        <param name="ExecuteOnComputer" comment="HostName" type="string"
            deprecated="true"/>
    </section>
</configvariables>
```

**Note**

Normally, the XML output produced by `ndb_config --configinfo --xml` is formatted using one line per element; we have added extra whitespace in the previous example, as well as the next one, for reasons of legibility. This should not make any difference to applications using this output, since most XML processors either ignore nonessential whitespace as a matter of course, or can be instructed to do so.

The XML output also indicates when changing a given parameter requires that data nodes be restarted using the `--initial` option. This is shown by the presence of an `initial="true"` attribute in the corresponding `<param>` element. In addition, the restart type (system or node) is also shown; if a given parameter requires a system restart, this is indicated by the presence of a `restart="system"` attribute in the corresponding `<param>` element. For example, changing the value set for the `Diskless` parameter requires a system initial restart, as shown here (with the `restart` and `initial` attributes highlighted for visibility):

```
<param name="Diskless" comment="Run wo/ disk" type="bool" default="false"
    restart="system" initial="true"/>
```

Currently, no `initial` attribute is included in the XML output for `<param>` elements corresponding to parameters which do not require initial restarts; in other words, `initial="false"` is the default, and the value `false` should be assumed if the attribute is not present. Similarly, the default restart type is `node` (that is, an online or “rolling” restart of the cluster), but the `restart` attribute is included only if the restart type is `system` (meaning that all cluster nodes must be shut down at the same time, then restarted).

Deprecated parameters are indicated in the XML output by the `deprecated` attribute, as shown here:

```
<param name="NoOfDiskPagesToDiskAfterRestartACC" comment="DiskCheckpointSpeed"
    deprecated="true"/>
```
In such cases, the comment refers to one or more parameters that supersede the deprecated parameter. Similarly to initial, the deprecated attribute is indicated only when the parameter is deprecated, with deprecated="true", and does not appear at all for parameters which are not deprecated. (Bug #21127135)

Parameters that are required are indicated with mandatory="true", as shown here:

```xml
<param name="NodeId"
    comment="Number identifying application node (mysqld(API))"
    type="unsigned" mandatory="true"
    min="1" max="255"/>
```

In much the same way that the initial or deprecated attribute is displayed only for a parameter that requires an intial restart or that is deprecated, the mandatory attribute is included only if the given parameter is actually required.

**Important**

The --xml option can be used only with the --configinfo option. Using --xml without --configinfo fails with an error.

Unlike the options used with this program to obtain current configuration data, --configinfo and --xml use information obtained from the NDB Cluster sources when ndb_config was compiled. For this reason, no connection to a running NDB Cluster or access to a config.ini or my.cnf file is required for these two options.

Combining other ndb_config options (such as --query or --type) with --configinfo (with or without the --xml option is not supported. Currently, if you attempt to do so, the usual result is that all other options besides --configinfo or --xml are simply ignored. However, this behavior is not guaranteed and is subject to change at any time. In addition, since ndb_config, when used with the --configinfo option, does not access the NDB Cluster or read any files, trying to specify additional options such as --ndb-connectstring or --config-file with --configinfo serves no purpose.

**Examples**

1. To obtain the node ID and type of each node in the cluster:

   ```shell```
   ./ndb_config --query=nodeid,type --fields=':' --rows='\n'
   1:ndbd
   2:ndbd
   3:ndbd
   4:ndbd
   5:ndb_mgmd
   6:mysqld
   7:mysqld
   8:mysqld
   9:mysqld
   ```

   In this example, we used the --fields options to separate the ID and type of each node with a colon character (:), and the --rows options to place the values for each node on a new line in the output.

2. To produce a connection string that can be used by data, SQL, and API nodes to connect to the management server:

   ```shell```
   ./ndb_config --config-file=usr/local/mysql/cluster-data/config.ini \  
   --query=hostname,portnumber --fields=: --rows=, --type=ndb_mgmd
   198.51.100.179:1186
   ```

3. This invocation of ndb_config checks only data nodes (using the --type option), and shows the values for each node's ID and host name, as well as the values set for its DataMemory and DataDir parameters:
In this example, we used the short options `-f` and `-r` for setting the field delimiter and row separator, respectively, as well as the short option `-q` to pass a list of parameters to be obtained.

4. To exclude results from any host except one in particular, use the `--host` option:

```
shell> ./ndb_config --host=198.51.100.176 -f :-r \n3: ndbd
5: ndb_mgmd
```

In this example, we also used the short form `-q` to determine the attributes to be queried.

Similarly, you can limit results to a node with a specific ID using the `--nodeid` option.

6.8 `ndb_delete_all` — Delete All Rows from an NDB Table

`ndb_delete_all` deletes all rows from the given NDB table. In some cases, this can be much faster than `DELETE` or even `TRUNCATE TABLE`.

**Usage**

```
ndb_delete_all -c connection_string tbl_name -d db_name
```

This deletes all rows from the table named `tbl_name` in the database named `db_name`. It is exactly equivalent to executing `TRUNCATE db_name.tbl_name` in MySQL.

The following table includes options that are specific to `ndb_delete_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_delete_all`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

**Table 6.7 Command-line options used with the program ndb_delete_all**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--database=dbname</code>, <code>-d</code></td>
<td>Name of the database in which the table is found</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--transactional</code>, <code>-t</code></td>
<td>Perform the delete in a single transaction (may run out of operations)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--tupscan</code></td>
<td>Run tup scan</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--diskscan</code></td>
<td>Run disk scan</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

* `--transactional`, `-t`

Use of this option causes the delete operation to be performed as a single transaction.

**Warning**

With very large tables, using this option may cause the number of operations available to the cluster to be exceeded.
Prior to NDB 8.0.18, 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.

## 6.9 ndb_desc — Describe NDB Tables

`ndb_desc` provides a detailed description of one or more NDB tables.

### Usage

```bash
ndb_desc -c connection_string tbl_name -d db_name [options]
ndb_desc -c connection_string index_name -d db_name -t tbl_name
```

Additional options that can be used with `ndb_desc` are listed later in this section.

### Sample Output

MySQL table creation and population statements:

```sql
USE test;
CREATE TABLE fish (
    id INT NOT NULL AUTO_INCREMENT,
    name VARCHAR(20) NOT NULL,
    length_mm INT NOT NULL,
    weight_gm INT NOT NULL,
    PRIMARY KEY pk (id),
    UNIQUE KEY uk (name)
) ENGINE=NDB;
```

Insert into `fish` values:

```sql
(NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),
(NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),
(NULL, 'grouper', 900, 125000), (NULL, 'puffer', 250, 250);
```

Output from `ndb_desc`:

```
shell> ./ndb_desc -c localhost fish -d test -p
-- fish --
Version: 2
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: 337
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 2
FragmentCount: 2
PartitionBalance: FOR_RP_BY_LDM
ExtraRowGciBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options:
HashMap: DEFAULT-HASHMAP-3840-2
-- Attributes --
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCREMENT
name Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT_VAR ST=MEMORY DYNAMIC
length_mm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
weight_gm Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
uk(name) - OrderedIndex
uk$unique(name) - UniqueHashIndex
-- Per partition info --
```
Information about multiple tables can be obtained in a single invocation of `ndb_desc` by using their names, separated by spaces. All of the tables must be in the same database.

You can obtain additional information about a specific index using the `--table` (short form: `--t`) option and supplying the name of the index as the first argument to `ndb_desc`, as shown here:

```
shell> ./ndb_desc uk --d test --t fish
```

When an index is specified in this way, the `--extra-partition-info` and `--extra-node-info` options have no effect.

The `Version` column in the output contains the table's schema object version. For information about interpreting this value, see NDB Schema Object Versions.

Three of the table properties that can be set using `NDB_TABLE` comments embedded in `CREATE TABLE` and `ALTER TABLE` statements are also visible in `ndb_desc` output. The table's `FRAGMENT_COUNT_TYPE` is always shown in the `FragmentCountType` column. `READ_ONLY` and `FULLY_REPLICATED`, if set to 1, are shown in the `Table options` column. You can see this after executing the following `ALTER TABLE` statement in the `mysql` client:

```
mysql> ALTER TABLE fish COMMENT='NDB_TABLE=READ_ONLY=1,FULLY_REPLICATED=1';
1 row in set, 1 warning (0.00 sec)
```

```
+---------+------+---------------------------------------------------------------------------------------------------------+
| Level   | Code | Message                                                                                                 |
|---------+------|---------------------------------------------------------------------------------------------------------|
| Warning | 1296 | Got error 4503 'Table property is FRAGMENT_COUNT_TYPE=ONE_PER_LDM_PER_NODE but not in comment' from NDB |
|---------+------|---------------------------------------------------------------------------------------------------------|
```
The warning is issued because \texttt{READ\_ONLY=1} requires that the table's fragment count type is (or be set to) \texttt{ONE\_PER\_LDM\_PER\_NODE\_GROUP}; NDB sets this automatically in such cases. You can check that the \texttt{ALTER TABLE} statement has the desired effect using \texttt{SHOW CREATE TABLE}:

\begin{verbatim}
mysql> SHOW CREATE TABLE fish
+------------------+
| Table: fish      |
+------------------+
| Create Table: CREATE TABLE `fish` ( |
| `id` int(11) NOT NULL AUTO_INCREMENT, |
| `name` varchar(20) NOT NULL, |
| `length_mm` int(11) NOT NULL, |
| `weight_gm` int(11) NOT NULL, |
| PRIMARY KEY (`id`), |
| UNIQUE KEY `uk` (`name`) |
| ) ENGINE=ndbcluster DEFAULT CHARSET=latin1 |
| COMMENT='NDB\_TABLE=READ\_BACKUP=1,FULLY\_REPLICATED=1' |
+------------------+
1 row in set (0.01 sec)
\end{verbatim}

Because \texttt{FRAGMENT\_COUNT\_TYPE} was not set explicitly, its value is not shown in the comment text printed by \texttt{SHOW CREATE TABLE}, however, displays the updated value for this attribute. The \texttt{Table options} column shows the binary properties just enabled. You can see this in the output shown here (emphasized text):

\begin{verbatim}
shell> ./ndb_desc --c localhost fish -- fish
Version: 4
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: 380
Max Rows: 0
Row Checksum: 1
Row GCI: 1
SingleUserMode: 0
ForceVarPart: 1
PartitionCount: 1
FragmentCount: 1
FragmentCountType: ONE\_PER\_LDM\_PER\_NODE\_GROUP
ExtraRowGcIBits: 0
ExtraRowAuthorBits: 0
TableStatus: Retrieved
Table options: readbackup, fullyreplicated
HashMap: DEFAULT-HASHMAP-3840-1
-- Attributes --
| id | Int | PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO\_INCR |
| name | Varchar(20;latin1_swedish_ci) NOT NULL AT=SHORT\_VAR ST=MEMORY DYNAMIC |
| length_mm | Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC |
| weight_gm | Int NOT NULL AT=FIXED ST=MEMORY DYNAMIC |
-- Indexes --
| PRIMARY KEY(id) | - UniqueHashIndex |
| PRIMARY(id) | - OrderedIndex |
| uk(name) | - OrderedIndex |
| uk$unique(name) | - UniqueHashIndex |
-- Per partition info --
| Partition | Row count | Commit count | Frag fixed memory | Frag varsized memory | Extent_space |
| NDBT\_ProgramExit: 0 - OK |
\end{verbatim}

For more information about these table properties, see Setting NDB\_TABLE Options.

The \texttt{Extent\_space} and \texttt{Free extent\_space} columns are applicable only to NDB tables having columns on disk; for tables having only in-memory columns, these columns always contain the value 0.

To illustrate their use, we modify the previous example. First, we must create the necessary Disk Data objects, as shown here:
CREATE LOGFILE GROUP lg_1
   ADD UNDOFILE 'undo_1.log'
   INITIAL_SIZE 16M
   UNDO_BUFFER_SIZE 2M
   ENGINE NDB;
ALTER LOGFILE GROUP lg_1
   ADD UNDOFILE 'undo_2.log'
   INITIAL_SIZE 12M
   ENGINE NDB;
CREATE TABLESPACE ts_1
   ADD DATAFILE 'data_1.dat'
   USE LOGFILE GROUP lg_1
   INITIAL_SIZE 32M
   ENGINE NDB;
ALTER TABLESPACE ts_1
   ADD DATAFILE 'data_2.dat'
   INITIAL_SIZE 48M
   ENGINE NDB;

(For more information on the statements just shown and the objects created by them, see
Section 7.10.1, "NDB Cluster Disk Data Objects", as well as CREATE LOGFILE GROUP Statement,
and CREATE TABLESPACE Statement.)

Now we can create and populate a version of the fish table that stores 2 of its columns on disk
(deleting the previous version of the table first, if it already exists):

DROP TABLE IF EXISTS fish;
CREATE TABLE fish (  
id INT NOT NULL AUTO_INCREMENT,  
ame VARCHAR(20) NOT NULL,  
length_mm INT NOT NULL,  
weight_gm INT NOT NULL,  
PRIMARY KEY pk (id),  
UNIQUE KEY uk (name)  ) TABLESPACE ts_1 STORAGE DISK  
ENGINE=NDB;
INSERT INTO fish VALUES  
(NULL, 'guppy', 35, 2), (NULL, 'tuna', 2500, 150000),  
(NULL, 'shark', 3000, 110000), (NULL, 'manta ray', 1500, 50000),  
(NULL, 'grouper', 900, 125000), (NULL ,'puffer', 250, 2500);

When run against this version of the table, ndb_desc displays the following output:

shell> ./ndb_desc --localhost fish --d test --p
-- fish --  
Version: 1  
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: 1001  
Max Rows: 0  
Row Checksum: 1  
Row GCI: 1  
SingleUserMode: 0  
ForceVarPart: 1  
PartitionCount: 2  
FragmentCount: 2  
PartitionBalance: FOR_RP_BY_LDM  
ExtraRowGciBits: 0  
ExtraRowAuthorBits: 0  
TableStatus: Retrieved  
Table options: readbackup  
Tablespace id: 16  
Tablespace: ts_1  
-- Attributes --  
id Int PRIMARY KEY DISTRIBUTION KEY AT=FIXED ST=MEMORY AUTO_INCR
This means that 1048576 bytes are allocated from the tablespace for this table on each partition, of which 1044440 bytes remain free for additional storage. In other words, 1048576 - 1044440 = 4136 bytes per partition is currently being used to store the data from this table's disk-based columns. The number of bytes shown as Free extent space is available for storing on-disk column data from the fish table only; for this reason, it is not visible when selecting from the INFORMATION_SCHEMA.FILES table.

Tablespace id and Tablespace are displayed for Disk Data tables beginning with NDB 8.0.21.

For fully replicated tables, ndb_desc shows only the nodes holding primary partition fragment replicas; nodes with copy fragment replicas (only) are ignored. You can obtain such information, using the mysql client, from the table_distribution_status, table_fragments, table_info, and table_replicas tables in the ndbinfo database.

The following table includes options that are specific to ndb_desc. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_desc), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.8 Command-line options used with the program ndb_desc

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--auto-inc, -a</td>
<td>Show next value for AUTO_INCREMENT column if table has one</td>
<td>ADDED: NDB 8.0.21</td>
</tr>
<tr>
<td>--blob-info, -b</td>
<td>Include partition information for BLOB tables in output. Requires</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td></td>
<td>that the -p option also be used</td>
<td></td>
</tr>
<tr>
<td>--context, -x</td>
<td>Show extra information for table such as database, schema, name, internal ID</td>
<td>ADDED: NDB 8.0.21</td>
</tr>
<tr>
<td>--database=dbname, -d</td>
<td>Name of database containing table</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--extra-node-info, -n</td>
<td>Include partition-to-data-node mappings in output. Requires that the -p option also be used</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--extra-partition-info, -p</td>
<td>Display information about partitions</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--retries=#, -r</td>
<td>Number of times to retry the connection (once per second)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--table=tbl_name, -t</td>
<td>Specify the table in which to find an index. When this option is</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
ndb_drop_index — Drop Index from an NDB Table

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--unqualified, -u</td>
<td>used, -p and -n have no effect and are ignored</td>
<td>Use unqualified table names</td>
</tr>
<tr>
<td></td>
<td>Use unqualified table names</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

- **--auto-inc, -a**
  Show the next value for a table's AUTO_INCREMENT column, if it has one.

- **--blob-info, -b**
  Include information about subordinate BLOB and TEXT columns.
  Use of this option also requires the use of the --extra-partition-info (-p) option.

- **--context, -x**
  Show additional contextual information for the table such as schema, database name, table name, and the table’s internal ID.

- **--database=db_name, -d**
  Specify the database in which the table should be found.

- **--extra-node-info, -n**
  Include information about the mappings between table partitions and the data nodes upon which they reside. This information can be useful for verifying distribution awareness mechanisms and supporting more efficient application access to the data stored in NDB Cluster.
  Use of this option also requires the use of the --extra-partition-info (-p) option.

- **--extra-partition-info, -p**
  Print additional information about the table’s partitions.

- **--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.

Table indexes listed in the output are ordered by ID.

6.10 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`.

The following table includes options that are specific to `ndb_drop_index`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_drop_index`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

**Table 6.9 Command-line options used with the program `ndb_drop_index`**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--database=dbname, -d</code></td>
<td>Name of database in which table is found</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

**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:*

```shell
shell> ./ndb_drop_index -c localhost dogs ix -d ctest1
Dropping index dogs/idx...OK
NDBT_ProgramExit: 0 - OK
shell> ./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.
Type 'help;' or '\h' for help. Type '\c' to clear the buffer.
mysql>
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 6.11, “ndb_drop_table — Drop an NDB Table”) to drop the table.

**6.11 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
```

The following table includes options that are specific to `ndb_drop_table`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_drop_table`),
**ndb_error_reporter** — NDB Error-Reporting Utility

see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.10 Command-line options used with the program ndb_drop_table

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--database=dbname,</td>
<td>Name of database in which table is found</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-d</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Prior to NDB 8.0.17, an NDB table dropped using this utility persisted in the MySQL data dictionary but could not be dropped using `DROP TABLE` in the `mysql` client. In NDB 8.0.17 and later, such “orphan” tables can be dropped using `DROP TABLE`. (Bug #29125206, Bug #93672)

**6.12 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.

The following table includes command options specific to the NDB Cluster program `ndb_error_reporter`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_error_reporter`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

Table 6.11 Command-line options used with the program `ndb_error_reporter`

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--connection-timeout=timeout</td>
<td>Number of seconds to wait when connecting to nodes before timing out</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--dry-scp</td>
<td>Disable scp with remote hosts; used only for testing</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--fs</td>
<td>Include file system data in error report; can use a large amount of disk space</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--skip-nodegroup=nodegroup_id</td>
<td>Skip all nodes in the node group having this ID</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

**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_YYYYMMDDhhmms.tar.bz2`, where `YYYYMMDDhhmms` is a datetime string.

`ndb_error_reporter` also accepts the options listed here:

- `--connection-timeout=timeout`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--connection-timeout=timeout</code></th>
<th>Type</th>
<th>Integer</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>0</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Wait this many seconds when trying to connect to nodes before timing out.

- **--dry-scp**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--dry-scp</td>
<td>Boolean</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

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

- **--fs**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--fs</td>
<td>Boolean</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  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**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Type</th>
<th>Default Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>--connection-timeout=timeout</td>
<td>Integer</td>
<td>0</td>
</tr>
</tbody>
</table>

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

## 6.13 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. `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:

```sql
mysql> USE myinnodb;
mysql> SELECT * INTO OUTFILE '/tmp/myndb_table.csv'
    > > FIELDS TERMINATED BY ',' OPTIONALLY ENCLOSED 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;
```

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:

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

The following table includes options that are specific to `ndb_import`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_import`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

**Table 6.12 Command-line options used with the program ndb_import**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--abort-on-error</code></td>
<td>Dump core on any fatal error; used for debugging</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--ai-increment=#</code></td>
<td>For table with hidden PK, specify autoincrement increment. See mysqld</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--ai-offset=#</code></td>
<td>For table with hidden PK, specify autoincrement offset. See mysqld</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--ai-prefetch-sz=#</code></td>
<td>For table with hidden PK, specify number of autoincrement values that are prefetched. See mysqld</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connections=#</code></td>
<td>Number of cluster connections to create</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--continue</code></td>
<td>When job fails, continue to next job</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--db-workers=#</code></td>
<td>Number of threads, per data node, executing database operations</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--errins-type=name</code></td>
<td>Error insert type, for testing purposes; use &quot;list&quot; to obtain all possible values</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
<td>Added, Deprecated, or Removed</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------</td>
</tr>
<tr>
<td><code>--errins-delay=#</code></td>
<td>Error insert delay in milliseconds; random variation is added</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-enclosed-by=char</code></td>
<td>Same as FIELDS ENCLOSED BY option for LOAD DATA statements. For CSV input this is same as using --fields-optionally-enclosed-by</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-escaped-by=name</code></td>
<td>Same as FIELDS ESCAPED BY option for LOAD DATA statements</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-optionally-enclosed-by=char</code></td>
<td>Same as FIELDS OPTIONALLY ENCLOSED BY option for LOAD DATA statements</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-terminated-by=char</code></td>
<td>Same as FIELDS TERMINATED BY option for LOAD DATA statements</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--idlesleep=#</code></td>
<td>Number of milliseconds to sleep waiting for more to do</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--idlespin=#</code></td>
<td>Number of times to re-try before idlesleep</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--ignore-lines=#</code></td>
<td>Ignore first # lines in input file. Used to skip a non-data header</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--input-type=name</code></td>
<td>Input type: random or csv</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--input-workers=#</code></td>
<td>Number of threads processing input. Must be 2 or more if --input-type is csv</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--keep-state</code></td>
<td>State files (except non-empty *.rej files) are normally removed on job completion. Using this option causes all state files to be preserved instead</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--lines-terminated-by=name</code></td>
<td>Same as LINES TERMINATED BY option for LOAD DATA statements</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--max-rows=#</code></td>
<td>Import only this number of input data rows; default is 0, which imports all rows</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--monitor=#</code></td>
<td>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</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--no-asynch</code></td>
<td>Run database operations as batches, in single transactions</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--no-hint</code></td>
<td>Do not use distribution key hint to select data node (TC)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
<td>Added, Deprecated, or Removed</td>
</tr>
<tr>
<td>--------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--opbatch=#</code></td>
<td>A db execution batch is a set of transactions and operations sent to NDB kernel. This option limits NDB operations (including blob operations) in a db execution batch. Therefore it also limits number of async transactions. Value 0 is not valid.</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--opbytes=#</code></td>
<td>Limit bytes in execution batch (default 0 = no limit)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--output-type=name</code></td>
<td>Output type: ndb is default, null used for testing</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--output-workers=#</code></td>
<td>Number of threads processing output or relaying database operations</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--pagesize=#</code></td>
<td>Align I/O buffers to given size</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--pagecnt=#</code></td>
<td>Size of I/O buffers as multiple of page size. CSV input worker allocates a double-sized buffer</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--polltimeout=#</code></td>
<td>Timeout per poll for completed asynchronous transactions; polling continues until all polls are completed, or error occurs</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--rejects=#</code></td>
<td>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</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--resume</code></td>
<td>If job aborted (temporary error, user interrupt), resume with rows not yet processed</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--rowbatch=#</code></td>
<td>Limit rows in row queues (default 0 = no limit); must be 1 or more if --input-type is random</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--rowbytes=#</code></td>
<td>Limit bytes in row queues (0 = no limit)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--state-dir=name</code></td>
<td>Where to write state files; current directory is default</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--stats</code></td>
<td>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</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--tempdelay=#</code></td>
<td>Number of milliseconds to sleep between temporary errors</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--temperrors=#</code></td>
<td>Number of times a transaction can fail due to a temporary error,</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--verbose,</td>
<td>Enable verbose output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-v</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**--abort-on-error**

- Command-Line Format: `--abort-on-error`
- Type: Boolean
- Default Value: FALSE

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.

**--connections=#**

- Command-Line Format: `--connections=#`
- Type: Integer
- Default Value: 296
- Minimum Value: 1
- Maximum Value: 4294967295

For a table with hidden primary key, specify the number of connections per execution batch; 0 means any temporary error is fatal. Such errors do not cause any rows to be written to .rej file.
### Usage

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--connections=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Number of cluster connections to create.

- **--continue**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--continue</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

When a job fails, continue to the next job.

- **--db-workers=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--db-workers=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>4</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--errins-type=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
<tr>
<td>Valid Values</td>
<td>stopjob</td>
</tr>
<tr>
<td></td>
<td>stopall</td>
</tr>
<tr>
<td></td>
<td>sighup</td>
</tr>
<tr>
<td></td>
<td>sigint</td>
</tr>
<tr>
<td></td>
<td>list</td>
</tr>
</tbody>
</table>

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

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

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--errins-delay=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1000</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
<tr>
<td>Unit</td>
<td>ms</td>
</tr>
</tbody>
</table>
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=name  
  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.

- **--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=**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--idlespin= #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Number of times to retry before sleeping.

• **--ignore-lines=**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ignore-lines= #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

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=**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--input-type=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>csv</td>
</tr>
<tr>
<td>Valid Values</td>
<td>random, csv</td>
</tr>
</tbody>
</table>

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

• **--input-workers=**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--input-workers= #</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>4</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

Set the number of threads processing input.

• **--keep-state**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--keep-state</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
</tbody>
</table>

By default, ndb_import removes all state files (except non-empty *.rej files) when it completes a job. Specify this option (no argument is required) to force the program to retain all state files instead.
• `--lines-terminated-by=name`

**Command-Line Format**

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

**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.

• `--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.

• `--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.

• `--no-asynch`

**Command-Line Format**

```
--no-asynch
```

**Type**

Boolean
Usage

<table>
<thead>
<tr>
<th>Default Value</th>
<th>FALSE</th>
</tr>
</thead>
</table>

Run database operations as batches, in single transactions.

- **--no-hint**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-hint</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

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

- **--opbatch=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--opbatch=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>256</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
<tr>
<td>Unit</td>
<td>bytes</td>
</tr>
</tbody>
</table>

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

- **--opbytes=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--opbytes=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
<tr>
<td>Unit</td>
<td>bytes</td>
</tr>
</tbody>
</table>

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

- **--output-type=name**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--output-type=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>ndb</td>
</tr>
<tr>
<td>Valid Values</td>
<td>null</td>
</tr>
</tbody>
</table>

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

- **--output-workers=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--output-workers=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>2</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
</tbody>
</table>
Usage

Maximum Value | 4294967295

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

• **--pagesize=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--pagesize=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>4096</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
<tr>
<td>Unit</td>
<td>bytes</td>
</tr>
</tbody>
</table>

Align I/O buffers to the given size.

• **--pagecnt=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--pagecnt=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>64</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

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

• **--polltimeout=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--polltimeout=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1000</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
<tr>
<td>Unit</td>
<td>ms</td>
</tr>
</tbody>
</table>

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

• **--rejects=#**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--rejects=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

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.
### Usage

- **--resume**

  **Command-Line Format**
  ```
  --resume
  ```

  **Type**
  Boolean

  **Default Value**
  `FALSE`

  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
  ```

  **Type**
  Boolean

  **Default Value**
  `false`

  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=name
  ```

  **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.

- **--tempdelay=#**

  **Command-Line Format**
  ```
  --tempdelay=#
  ```
### 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:

```
shell> ndb_index_stat -d test mytable
table:City index:PRIMARY fragCount:2
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:

\[ \text{keyBytes} = \text{sampleCount} \times (4 \text{ bytes per index}) = 1994 \times 4 = 7976 \]

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:

```
shell> 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
shell>
```

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. For options common to most NDB Cluster programs (including ndb_index_stat), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

**Table 6.13 Command-line options used with the program ndb_index_stat**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--database=name, -d --delete</td>
<td>Name of database containing table Delete index statistics for table, stopping any auto-update previously configured</td>
<td>(Supported in all NDB releases based on MySQL 8.0) (Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### ndb_index_stat statistics options

The following options 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=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.

- **--delete**

  **Command-Line Format**: `--delete`

---

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--update</td>
<td>Update index statistics for table, restarting any auto-update previously configured</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--dump</td>
<td>Print query cache</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--query=#</td>
<td>Perform random range queries on first key attr (must be int unsigned)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-drop</td>
<td>Drop any statistics tables and events in NDB kernel (all statistics are lost)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-create</td>
<td>Create all statistics tables and events in NDB kernel, if none of them already exist</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-create-if-not-exist</td>
<td>Create any statistics tables and events in NDB kernel that do not already exist</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-create-if-not-valid</td>
<td>Create any statistics tables or events that do not already exist in the NDB kernel, after dropping any that are invalid</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-check</td>
<td>Verify that NDB system index statistics and event tables exist</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-skip-tables</td>
<td>Do not apply sys-* options to tables</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sys-skip-events</td>
<td>Do not apply sys-* options to events</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Turn on verbose output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--loops=#</td>
<td>Set the number of times to perform given command; default is 0</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### Options

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
<tr>
<td>Minimum Value</td>
<td></td>
</tr>
<tr>
<td>Maximum Value</td>
<td></td>
</tr>
</tbody>
</table>

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

- **--update**
  
  **Command-Line Format**  
  **--update**

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
<tr>
<td>Minimum Value</td>
<td></td>
</tr>
<tr>
<td>Maximum Value</td>
<td></td>
</tr>
</tbody>
</table>

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

- **--dump**
  
  **Command-Line Format**  
  **--dump**

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
<tr>
<td>Minimum Value</td>
<td></td>
</tr>
<tr>
<td>Maximum Value</td>
<td></td>
</tr>
</tbody>
</table>

Dump the contents of the query cache.

- **--query=#**
  
  **Command-Line Format**  
  **--query=#**

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>MAX_INT</td>
</tr>
</tbody>
</table>

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

**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**
  
  **Command-Line Format**  
  **--sys-drop**

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>false</td>
</tr>
<tr>
<td>Minimum Value</td>
<td></td>
</tr>
<tr>
<td>Maximum Value</td>
<td></td>
</tr>
</tbody>
</table>

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

- **--sys-create**

  **Command-Line Format**
  
  --sys-create

  **Type**
  
  Boolean

  **Default Value**
  
  false

  **Minimum Value**
  
  
  **Maximum Value**
  
  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

  **Type**
  
  Boolean

  **Default Value**
  
  false

  **Minimum Value**
  
  
  **Maximum Value**
  
  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

  **Type**
  
  Boolean

  **Default Value**
  
  false

  **Minimum Value**
  
  
  **Maximum Value**
  
  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

  **Type**
  
  Boolean

  **Default Value**
  
  false

  **Minimum Value**
  
  
  **Maximum Value**
  
  Verify that all required system statistics tables and events exist in the NDB kernel.

- **--sys-skip-tables**

  **Command-Line Format**
  
  --sys-skip-tables

  **Type**
  
  Boolean

  **Default Value**
  
  false

  **Minimum Value**
  
  
  **Maximum Value**
  

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

- **--sys-skip-events**
  
  | Command-Line Format | --sys-skip-events |
  | Type              | Boolean          |
  | Default Value     | false            |
  | Minimum Value     |                 |
  | Maximum Value     |                 |

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

- **--verbose**
  
  | Command-Line Format | --verbose |
  | Type               | Boolean   |
  | Default Value      | false     |
  | Minimum Value      |           |
  | Maximum Value      |           |

Turn on verbose output.

- **--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).

### 6.15 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
```

The following table includes options that are specific to **ndb_move_data**. Additional descriptions follow the table. For options common to most NDB Cluster programs (including **ndb_move_data**), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

**Table 6.14 Command-line options used with the program ndb_move_data**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--abort-on-error</strong></td>
<td>Dump core on permanent error (debug option)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### Usage

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--character-sets-dir=name</td>
<td>Directory where character sets are</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--database=dbname, -d</td>
<td>Name of database in which table is found</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--drop-source</td>
<td>Drop source table after all rows have been moved</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--error-insert</td>
<td>Insert random temporary errors (testing option)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--exclude-missing-columns</td>
<td>Ignore extra columns in source or target table</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--lossy-conversions, -l</td>
<td>Allow attribute data to be truncated when converted to smaller type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--promote-attributes, -A</td>
<td>Allow attribute data to be converted to larger type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--staging-tries=x[,y[,z]]</td>
<td>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)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose</td>
<td>Enable verbose messages</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

* **--abort-on-error**

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

Type: Boolean

Default Value: `FALSE`

Dump core on permanent error (debug option).

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

Command-Line Format: `--character-sets-dir=name`

Type: String

Default Value: `[none]`

Directory where character sets are.

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

Command-Line Format: `--database=dbname`

Type: String

Default Value: `TEST_DB`

Name of the database in which the table is found.

* **--drop-source**

Command-Line Format: `--drop-source`
### Usage

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Drop source table after all rows have been moved.</td>
<td></td>
<td><strong>FALSE</strong></td>
<td><strong>--error-insert</strong>&lt;br&gt;Insert random temporary errors (testing option).</td>
</tr>
<tr>
<td><strong>--exclude-missing-columns</strong>&lt;br&gt;Ignore extra columns in source or target table.</td>
<td></td>
<td><strong>FALSE</strong></td>
<td><strong>--lossy-conversions,-l</strong>&lt;br&gt;Allow attribute data to be truncated when converted to a smaller type.</td>
</tr>
<tr>
<td><strong>--promote-attributes,-A</strong>&lt;br&gt;Allow attribute data to be converted to a larger type.</td>
<td></td>
<td><strong>FALSE</strong></td>
<td><strong>--staging-tries=x[,y[,z]]</strong>&lt;br&gt;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).</td>
</tr>
<tr>
<td><strong>--verbose</strong>&lt;br&gt;Enable verbose messages.</td>
<td></td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
</tbody>
</table>

---

#### Command-Line Format

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--error-insert</strong></td>
<td>Boolean</td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--exclude-missing-columns</strong></td>
<td></td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--lossy-conversions</strong></td>
<td></td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--promote-attributes</strong></td>
<td></td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--staging-tries=x[,y[,z]]</strong></td>
<td></td>
<td><strong>0,1000,60000</strong></td>
<td></td>
</tr>
<tr>
<td><strong>--verbose</strong></td>
<td></td>
<td><strong>FALSE</strong></td>
<td></td>
</tr>
</tbody>
</table>
6.16 **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:

```
shell> 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:

```
shell> ndb perror -s 323
Invalid nodegroup id, nodegroup already existing: Permanent error: Application error
```

Like **perror**, **ndb perror** accepts multiple error codes:

```
shell> 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 6.15 Command-line options used with the program ndb perror**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>--help, -?</strong></td>
<td>Display help text</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><strong>--ndb</strong></td>
<td>For compatibility with applications depending on old versions of perror; does nothing</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><strong>--silent, -s</strong></td>
<td>Show error message only</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><strong>--version, -V</strong></td>
<td>Print program version information and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><strong>--verbose, -v</strong></td>
<td>Verbose output; disable with <strong>--silent</strong></td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

**Additional Options**

- **--help, -?**
ndb_print_backup_file — Print NDB Backup File Contents

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--backup-password=string, -P</td>
<td>Use this password to decrypt file</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td>--backup-password-from-stdin[=value]</td>
<td>Get decryption password in a secure fashion from STDIN</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
</tbody>
</table>

ndb_print_backup_file obtains diagnostic information from a cluster backup file.

Table 6.16 Command-line options used with the program ndb_print_backup_file

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>Type</th>
<th>Default Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>--help</td>
<td>Boolean</td>
<td>TRUE</td>
<td>Display program help text and exit.</td>
</tr>
<tr>
<td>--ndb</td>
<td>Boolean</td>
<td>TRUE</td>
<td>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.</td>
</tr>
<tr>
<td>--silent, -s</td>
<td>Boolean</td>
<td>TRUE</td>
<td>Show error message only.</td>
</tr>
<tr>
<td>--version, -V</td>
<td>Boolean</td>
<td>TRUE</td>
<td>Print program version information and exit.</td>
</tr>
<tr>
<td>--verbose, -v</td>
<td>Boolean</td>
<td>TRUE</td>
<td>Verbose output; disable with --silent.</td>
</tr>
</tbody>
</table>
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 7.8.1, “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 8.0.17 and later, this program can also be used to read undo log files.

---

**Format** | **Description** | **Added, Deprecated, or Removed**
--- | --- | ---
--control-directory-number=##, -c | Control directory number | ADDED: NDB 8.0.24
--fragment-id=##, -f | Fragment ID | ADDED: NDB 8.0.24
--help, -h, -? | Print usage information | ADDED: NDB 8.0.24
--no-print-rows, -u | Do not print rows | ADDED: NDB 8.0.24
--print-header-words, -h | Print header words | ADDED: NDB 8.0.24
--print-restored-rows | Print restored rows | ADDED: NDB 8.0.24
--print-rows, -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, -n | File containing row ID to check for | ADDED: NDB 8.0.24
--show-ignored-rows, -i | Show ignored rows | ADDED: NDB 8.0.24
--table-id, -t | Table ID | ADDED: NDB 8.0.24
--verbose[=#], -v | Verbosity level | ADDED: NDB 8.0.24
Options

Prior to NDB 8.0.24 and later, `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.

1. **--backup-password**
   - Command-Line Format: `--backup-password=string`
   - 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.

2. **--backup-password-from-stdin**
   - Command-Line Format: `--backup-password-from-stdin=[value]`
   - Introduced: 8.0.24-ndb-8.0.24
   - Type: Boolean
   - Default Value: `FALSE`

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

3. **--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`.

4. **--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`.

5. **--help**
   - Command-Line Format: `--help`
   - Introduced: 8.0.24-ndb-8.0.24
   - Type: Boolean
   - Default Value

   Print program usage information.
Options

- **--no-print-rows**

  **Command-Line Format**
  --no-print-rows

  **Introduced**
  8.0.24-ndb-8.0.24

  **Type**
  Boolean

  **Default Value**
  FALSE

  Do not include rows in output.

- **--print-header-words**

  **Command-Line Format**
  --print-header-words

  **Introduced**
  8.0.24-ndb-8.0.24

  **Type**
  Boolean

  **Default Value**
  FALSE

  Include header words in output.

- **--print-restored-rows**

  **Command-Line Format**
  --print-restored-rows

  **Introduced**
  8.0.24-ndb-8.0.24

  **Type**
  Boolean

  **Default Value**
  FALSE

  Include restored rows in output, using the file `LCP/c/TtFf.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

  **Type**
  Boolean

  **Default Value**
  TRUE

  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

  **Type**
  Boolean

  **Default Value**
  FALSE

  Print rows per page.
• **--rowid-file**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--rowid-file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.24-ndb-8.0.24</td>
</tr>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

File to check for row ID.

• **--show-ignored-rows**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--show-ignored-rows</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.24-ndb-8.0.24</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

Show ignored rows.

• **--table-id**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--table-id</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.24-ndb-8.0.24</td>
</tr>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

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

• **--verbose**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--verbose[#]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.24-ndb-8.0.24</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
</tbody>
</table>

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

### 6.18 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.
Additional Options

`ndb_print_file` supports the following options:

- `-v`: Make output verbose.
- `-q`: Supress output (quiet mode).
- `--help`, `-h`, `-?`: Print help message.

For more information, see Section 7.10, "NDB Cluster Disk Data Tables".

6.19 `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 `SX.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

```
shell> 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
------- Fragment with FragId: 0 -------
Preferred Primary: 2 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
------- Stored Replica -------
Replica node is: 2 initialGci: 2 maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 0 lcpStatus: invalid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
------- Fragment with FragId: 1 -------
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
------- Stored Replica -------
Replica node is: 3 initialGci: 2 maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 0 lcpStatus: invalid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
------- Fragment with FragId: 1 -------
Preferred Primary: 3 numStoredReplicas: 2 numOldStoredReplicas: 0 distKey: 0 LogPartId: 0
------- Stored Replica -------
Replica node is: 3 initialGci: 2 maxGciCompleted: 1 maxGciStarted: 2 lcpId: 1 lcpStatus: valid
LcpNo[0]: maxGciCompleted: 1 maxGciStarted: 2 lcpId: 0 lcpStatus: invalid
LcpNo[1]: maxGciCompleted: 0 maxGciStarted: 0 lcpId: 0 lcpStatus: invalid
```
6.20 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.

6.21 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.

6.22 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`.

The following table includes options that are specific to the NDB Cluster program `ndb_redo_log_reader`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_redo_log_reader`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-dump</code></td>
<td>Print dump info</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-filedescriptors</code></td>
<td>Print file descriptors only</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--help</code></td>
<td>Print usage information</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-lap</code></td>
<td>Provide lap info, with max GCI started and completed</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-mbyte #</code></td>
<td>Starting megabyte</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-mbyteheaders</code></td>
<td>Show only first page header of each megabyte in file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-nocheck</code></td>
<td>Do not check records for errors</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-noprint</code></td>
<td>Do not print records</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-page #</code></td>
<td>Start with this page</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-pageheaders</code></td>
<td>Show page headers only</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-pageindex #</code></td>
<td>Start with this page index</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-twiddle</code></td>
<td>Bit-shifted dump</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

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:
### Usage

- **–dump**
  - **Command-Line Format**: `–dump`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - Print dump info.

- **–filedescriptors**
  - **Command-Line Format**: `–filedescriptors`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - `-filedescriptors`: Print file descriptors only.

- **--help**
  - **Command-Line Format**: `--help`
  - **--help**: Print usage information.

- **–lap**
  - **Command-Line Format**: `–lap`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - Provide lap info, with max GCI started and completed.

- **–mbyte #**
  - **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.

- **–mbyteheaders**
  - **Command-Line Format**: `–mbyteheaders`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - `-mbyteheaders`: Show only the first page header of every megabyte in the file.

- **–noprint**
  - **Command-Line Format**: `–noprint`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - `-noprint`: Do not print the contents of the log file.

- **–nocheck**
  - **Command-Line Format**: `–nocheck`
  - **Type**: Boolean
  - **Default Value**: FALSE
  - `-nocheck`: Do not check the log file for errors.
### 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.

#### Command-Line Format

<table>
<thead>
<tr>
<th>Option</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>-page #</code></td>
<td>Start at this page. # is an integer in the range 0 to 31, inclusive.</td>
</tr>
<tr>
<td><code>-pageheaders</code></td>
<td>Show page headers only.</td>
</tr>
<tr>
<td><code>-pageindex #</code></td>
<td>Start at this page index. # is an integer between 12 and 8191, inclusive.</td>
</tr>
<tr>
<td><code>-twiddle</code></td>
<td>Bit-shifted dump.</td>
</tr>
</tbody>
</table>

**Note**

Beginning with NDB 8.0.17, this program no longer prints `NDBT_ProgramExit: ...` when it finishes its run. Applications depending on this behavior should be modified accordingly when upgrading from NDB 8.0.16 or earlier to a NDB 8.0 later release.

**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 7.8.2, “Using The NDB Cluster Backup Files”).

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.
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 7.6, “NDB Cluster Single User Mode”, for more information.

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_restore. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_restore), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--allow-pk-changes=[0</td>
<td>1]</td>
<td>Allow changes to set of columns making up table’s primary key</td>
</tr>
<tr>
<td>--append</td>
<td>Append data to tab-delimited file</td>
<td></td>
</tr>
<tr>
<td>--backup-password=string</td>
<td>Supply a password for decrypting an encrypted backup with --decrypt; see documentation for allowed values</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td>--backup-password-from-stdin=[value]</td>
<td>Get decryption password in a secure fashion from STDIN; use together with --decrypt option</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td>--backup-path=dir_name</td>
<td>Path to backup files directory</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--backupid=#, -b</td>
<td>Restore from backup having this ID</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--connect, -c</td>
<td>Alias for --connectstring</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--decrypt</td>
<td>Decrypt an encrypted backup; requires --backup-password</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td>--disable-indexes</td>
<td>Causes indexes from backup to be ignored; may decrease time needed to restore data</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--dont-ignore-systab-0, -f</td>
<td>Do not ignore system table during restore; experimental only; not for production use</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--exclude-databases=db-list</td>
<td>List of one or more databases to exclude (includes those not named)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--exclude-intermediate-sql-tables=[TRUE</td>
<td>FALSE]</td>
<td>If TRUE (default), do not restore any intermediate tables (having names prefixed with '#sql-') that were left over from copying ALTER TABLE operations</td>
</tr>
<tr>
<td>--exclude-missing-columns</td>
<td>Causes columns from backup version of table that are missing</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
<td>Added, Deprecated, or Removed</td>
</tr>
<tr>
<td>---------------------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------</td>
</tr>
<tr>
<td><code>--exclude-missing-tables</code></td>
<td>Causes tables from backup that are missing from database to be ignored</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--exclude-tables=table-list</code></td>
<td>List of one or more tables to exclude (includes those in same database that are not named); each table reference must include database name</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-enclosed-by=char</code></td>
<td>Fields are enclosed by this character</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--fields-terminated-by=char</code></td>
<td>Fields are terminated by this character</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--hex</code></td>
<td>Print binary types in hexadecimal format</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>`--ignore-extended-pk-updates[=0</td>
<td>1]`</td>
<td>Ignore log entries containing updates to columns now included in extended primary key</td>
</tr>
<tr>
<td><code>--include-databases=db-list</code></td>
<td>List of one or more databases to restore (excludes those not named)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--include-stored-grants</code></td>
<td>Restore shared users and grants to <code>ndb_sql_metadata</code> table</td>
<td>ADDED: NDB 8.0.19</td>
</tr>
<tr>
<td><code>--include-tables=table-list</code></td>
<td>List of one or more tables to restore (excludes those in same database that are not named); each table reference must include database name</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--lines-terminated-by=char</code></td>
<td>Lines are terminated by this character</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--lossy-conversions, -L</code></td>
<td>Allow lossy conversions of column values (type demotions or changes in sign) when restoring data from backup</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--no-binlog</code></td>
<td>If mysqld is connected and using binary logging, do not log restored data</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--no-restore-disk-objects</code></td>
<td>Do not restore objects relating to Disk Data</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>-d</code></td>
<td>Do not upgrade array type for varsize attributes which do not already resize VAR data, and do not change column attributes</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--ndb-nodegroup-map=map</code></td>
<td>Nodegroup map for NDBCLUSTER storage engine;</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### ndb_restore

- **ndb_restore** — Restore an NDB Cluster Backup

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>-z</td>
<td>syntax: list of (source_nodegroup, destination_nodegroup)</td>
<td></td>
</tr>
<tr>
<td>--nodeid=#,</td>
<td>ID of node where backup was taken</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-n</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--num-slices=#</td>
<td>Number of slices to apply when restoring by slice</td>
<td>ADDED: NDB 8.0.20</td>
</tr>
<tr>
<td>--parallelism=#,</td>
<td>Number of parallel transactions to use while restoring data</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--preserve-trailing-spaces,</td>
<td>Allow preservation of trailing spaces (including padding) when promoting fixed-width string types to variable-width types</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-P</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--print</td>
<td>Print metadata, data, and log to stdout (equivalent to --print-meta --print-data --print-log)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--print-data</td>
<td>Print data to stdout</td>
<td></td>
</tr>
<tr>
<td>--print-log</td>
<td>Print log to stdout</td>
<td></td>
</tr>
<tr>
<td>--print-meta</td>
<td>Print metadata to stdout</td>
<td></td>
</tr>
<tr>
<td>--print-sql-log</td>
<td>Write SQL log to stdout; default is FALSE</td>
<td></td>
</tr>
<tr>
<td>--progress-frequency=#</td>
<td>Print status of restore each given number of seconds</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--promote-attributes,</td>
<td>Allow attributes to be promoted when restoring data from backup</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-A</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--rebuild-indexes</td>
<td>Causes multithreaded rebuilding of ordered indexes found in backup; number of threads used is determined by setting BuildIndexThreads</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--remap-column=[db].[tbl].[col]:[fn]:[args]</td>
<td>Apply offset to value of specified column using indicated function and arguments</td>
<td>ADDED: NDB 8.0.21</td>
</tr>
<tr>
<td>--restore-data,</td>
<td>Restore table data and logs into NDB Cluster using NDB API</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-r</td>
<td>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</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--restore-epoch,</td>
<td></td>
<td></td>
</tr>
<tr>
<td>-e</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--restore-meta,</td>
<td>Restore metadata to NDB Cluster using NDB API</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-m</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
ndb_restore — Restore an NDB Cluster Backup

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--restore-privilege-tables</td>
<td>Restore MySQL privilege tables that were previously moved to NDB</td>
<td>DEPRECATED: NDB 8.0.16</td>
</tr>
<tr>
<td>--rewrite-database=olddb,newdb</td>
<td>Restore to differently named database</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--skip-broken-objects</td>
<td>Ignore missing blob tables in backup file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--skip-table-check,-s</td>
<td>Skip table structure check during restore</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--skip-unknown-objects</td>
<td>Causes schema objects not recognized by ndb_restore to be ignored when restoring backup made from newer NDB version to older version</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--slice-id=#</td>
<td>Slice ID, when restoring by slices</td>
<td>ADDED: NDB 8.0.20</td>
</tr>
<tr>
<td>--tab=dir_name,-T dir_name</td>
<td>Creates a tab-separated .txt file for each table in path provided</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--verbose=#</td>
<td>Level of verbosity in output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</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.

Prior to NDB 8.0.19, when ndb_restore was used to restore any tables containing unique indexes, it was necessary to include --disable-indexes or --rebuild-indexes. Beginning with NDB 8.0.19, when automatic metadata synchronization is enabled, this is no longer necessary.

The -c option is used to specify a connection string which tells ndb_restore where to locate the cluster management server (see Section 5.3.3, “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 5.3.7, “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:

```
shell> ndb_mgm -e "SHOW"
```

More detailed information about all options used by ndb_restore can be found in the following list:

- --allow-pk-changes
**ndb_restore** — Restore an NDB Cluster Backup

---

| 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 \texttt{ndb_restore} instances running in parallel. (Running \texttt{ndb_restore --rebuild-indexes} at the end of the restore process causes this index to be dropped.)

- \texttt{--append}

  Command-Line Format \hspace{1cm} \texttt{--append}

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

- \texttt{--backup-path=dir_name}

  Command-Line Format \hspace{1cm} \texttt{--backup-path=dir_name}

  Type \hspace{1cm} Directory name

  Default Value \hspace{1cm} ./

  The path to the backup directory is required; this is supplied to \texttt{ndb_restore} using the \texttt{--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 \texttt{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 \texttt{ndb_restore} executable is located, and may be optionally prefixed with \texttt{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 \texttt{ndb_restore} must be run twice—once for each storage node in the cluster where the backup was taken. However, \texttt{ndb_restore} cannot always restore backups made from a cluster running one version of MySQL to a cluster running a different MySQL version.

  \begin{mdframed}[backgroundcolor=yellow!25]
  \textbf{Important}
  
  It is not possible to restore a backup made from a newer version of NDB Cluster using an older version of \texttt{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 \texttt{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 7.5.22 to a cluster running NDB Cluster 7.4.32, you must use the \texttt{ndb_restore} that comes with the NDB Cluster 7.5.22 distribution.
  \end{mdframed}

  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 \texttt{config.ini} file available for each concurrent \texttt{ndb_restore} process. However, the data files must always be applied before the logs.

- \texttt{--backup-password=password}

  Command-Line Format \hspace{1cm} \texttt{--backup-password=string}

  Introduced \hspace{1cm} 8.0.22-ndb-8.0.22

  Type \hspace{1cm} String
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 can be up 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]
```

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 7.8.2, “Using The NDB Cluster Management Client to Create a Backup”.)

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

Use of this option is required 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.

```
--connect, -c
```

```
--connect
```

Alias for `--ndb-connectstring`.

```
--decrypt
```

```
--decrypt
```
ndb_restore — Restore an NDB Cluster Backup

<table>
<thead>
<tr>
<th>Introduced</th>
<th>8.0.22-ndb-8.0.22</th>
</tr>
</thead>
<tbody>
<tr>
<td>Decrypt an encrypted backup using the password supplied by the --backup-password option.</td>
<td></td>
</tr>
</tbody>
</table>

- **--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.

- **--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=db-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 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**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--exclude-tables=table-list</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Type</strong></td>
<td>String</td>
</tr>
<tr>
<td><strong>Default Value</strong></td>
<td>List of one or more tables to exclude; each table reference must include the database name. Often used together with <strong>--exclude-databases</strong>.</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--exclude-databases=db1</code></td>
<td>All tables in all databases except <code>db1</code> are restored; no tables in <code>db1</code> are restored</td>
</tr>
<tr>
<td><code>--exclude-databases=db1,db2 (or --exclude-databases=db1 --exclude-databases=db2)</code></td>
<td>All tables in all databases except <code>db1</code> and <code>db2</code> are restored; no tables in <code>db1</code> or <code>db2</code> are restored</td>
</tr>
<tr>
<td><code>--exclude-tables=db1.t1</code></td>
<td>All tables except <code>t1</code> in database <code>db1</code> are restored; all other tables in <code>db1</code> are restored; all tables in all other databases are restored</td>
</tr>
<tr>
<td><code>--exclude-tables=db1.t2,db2.t1 (or --exclude-tables=db1.t2 --exclude-tables=db2.t1)</code></td>
<td>All tables in database <code>db1</code> except for <code>t2</code> and all tables in database <code>db2</code> except for table <code>t1</code> are restored; no other tables in <code>db1</code> or <code>db2</code> are restored</td>
</tr>
</tbody>
</table>
ndb_restore — Restore an NDB Cluster Backup

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>restored; all tables in all other databases are restored</td>
</tr>
</tbody>
</table>

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:

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

- **--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\[=0|1\]
```

<table>
<thead>
<tr>
<th>Introduced</th>
<th>8.0.21-ndb-8.0.21</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>0</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1</td>
</tr>
</tbody>
</table>

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=db-list
```

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
</table>

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
```

<table>
<thead>
<tr>
<th>Introduced</th>
<th>8.0.19-ndb-8.0.19</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
</tbody>
</table>
In NDB 8.0.19 and later, \texttt{ndb_restore} does not by default restore shared users and grants (see Section 7.12, “Distributed MySQL Privileges with NDB\textunderscore STORED\_USER”) to the \texttt{ndb\_sql\_metadata} table. Specifying this option causes it to do so.

- \texttt{--include-tables=table-list}

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>\texttt{--include-tables=table-list}</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td></td>
</tr>
</tbody>
</table>

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

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

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

Table 6.20 Several invocations of \texttt{ndb_restore} using \texttt{--include-*} options, and their effects on restoring from an NDB Cluster backup.

<table>
<thead>
<tr>
<th>Option</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>\texttt{--include-databases=db1}</td>
<td>Only tables in database \texttt{db1} are restored; all tables in all other databases are ignored</td>
</tr>
<tr>
<td>\texttt{--include-databases=db1,db2} (or \texttt{--include-databases=db1 --include-databases=db2})</td>
<td>Only tables in databases \texttt{db1} and \texttt{db2} are restored; all tables in all other databases are ignored</td>
</tr>
<tr>
<td>\texttt{--include-tables=db1.t1}</td>
<td>Only table \texttt{t1} in database \texttt{db1} is restored; no other tables in \texttt{db1} or in any other database are restored</td>
</tr>
<tr>
<td>\texttt{--include-tables=db1.t2,db2.t1} (or \texttt{--include-tables=db1.t2 --include-tables=db2.t1})</td>
<td>Only the table \texttt{t2} in database \texttt{db1} and the table \texttt{t1} in database \texttt{db2} are restored; no other tables in \texttt{db1}, \texttt{db2}, or any other database are restored</td>
</tr>
</tbody>
</table>

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

```
shell> 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 \texttt{--include-*} (or \texttt{--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**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--lines-terminated-by=char</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>\n (linebreak)</td>
</tr>
</tbody>
</table>

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

• **--lossy-conversions, -L**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--lossy-conversions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE (If option is not used)</td>
</tr>
</tbody>
</table>

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.

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

• **--no-binlog**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-binlog</th>
</tr>
</thead>
</table>

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

• **--no-restore-disk-objects, -d**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-restore-disk-objects</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

This option stops **ndb_restore** from restoring any NDB Cluster Disk Data objects, such as tablespaces and log file groups; see Section 7.10, “NDB Cluster Disk Data Tables”, for more information about these.

• **--no-upgrade, -u**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-upgrade</th>
</tr>
</thead>
</table>

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-nodegroup-map=map, -z**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ndb-nodegroup-map=map</th>
</tr>
</thead>
</table>

This option can be used to restore a backup taken from one node group to a different node group. Its argument is a list of the form source_node_group, target_node_group.
--nodeid=#, -n

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--nodeid=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
<tr>
<td>Default Value</td>
<td>none</td>
</tr>
</tbody>
</table>

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 6.23.2, “Restoring to a different number of data nodes”, for additional information and examples.

In NDB 8.0.15 and later, this option is required.

--num-slices=#

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--num-slices=</th>
</tr>
</thead>
<tbody>
<tr>
<td>Introduced</td>
<td>8.0.20-ndb-8.0.20</td>
</tr>
<tr>
<td>Type</td>
<td>Integer</td>
</tr>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>1024</td>
</tr>
</tbody>
</table>

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:**

   ```shell
   shell> 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:**

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

   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:**

   ```shell
   shell> 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:**

   ```shell
   shell> 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`

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--parallelism=#</code></th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
</tbody>
</table>
ndb_restore — Restore an NDB Cluster Backup

| 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

Type | Boolean
Default Value | FALSE

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

Type | Boolean
Default Value | FALSE

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.
ndb_restore — Restore an NDB Cluster Backup

- `--print-log`  
  Command-Line Format: `--print-log`  
  Type: Boolean  
  Default Value: FALSE  
  Cause ndb_restore to output its log to stdin.

- `--print-meta`  
  Command-Line Format: `--print-meta`  
  Type: Boolean  
  Default Value: FALSE  
  Print all metadata to stdin.

- `--print-sql-log`  
  Command-Line Format: `--print-sql-log`  
  Type: Boolean  
  Default Value: FALSE  
  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.

  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

339
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$STnode_id`. These tables are not needed afterwards, and are normally deleted by `ndb_restore` following a successful restoration.

* `--rebuild-indexes`

**Command-Line Format**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th><code>--rebuild-indexes</code></th>
</tr>
</thead>
</table>

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**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>`--remap-column=[db].[tbl].[col]:[fn]::[args]</th>
</tr>
</thead>
</table>

Introduced 8.0.21-ndb-8.0.21

Type String
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 `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 7.8.2, “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:

```
shell> 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:

```
shell> 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**
ndb_restore — Restore an NDB Cluster Backup

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

**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 8.9, "NDB Cluster Backups With NDB Cluster Replication").

- **--restore-meta, -m**

**Command-Line Format**: **--restore-meta**

**Type**: Boolean

**Default Value**: FALSE

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 8.0.16 or later, 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

**Type**: Boolean

342
ndb_restore — Restore an NDB Cluster Backup

| Default Value | FALSE (If option is not used) |

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 8.0.16 and later, 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 4.7, “Upgrading and Downgrading NDB Cluster”, as well as Grant Tables.

- **--rewrite-database=olddb,newdb**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--rewrite-database=olddb,newdb</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>none</td>
</tr>
</tbody>
</table>

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):

```shell
shell> 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**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--skip-broken-objects</th>
</tr>
</thead>
</table>

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.
**ndb_restore** — Restore an NDB Cluster Backup

- **--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=dir_name`

  **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.

- **--verbose=#**

  **Command-Line Format**

  `--verbose=#`
### 6.23.1 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 4.7, “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 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.

#### 6.23.1.1 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:

- 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.

- 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 8.0.14 or later cannot be restored using `ndb_restore` to 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 8.0.14 or later of tables which were created in NDB 8.0.13 or earlier, and which have not been altered since upgrading to NDB 8.0.14 or later, 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.

---

<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>1</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>0</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>255</td>
</tr>
</tbody>
</table>

Sets the level for the verbosity of the output. The minimum is 0; the maximum is 255. The default value is 1.

**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.
• Encrypted backups created in NDB 8.0.22 and later cannot be restored using ndb_restore from NDB 8.0.21 or earlier.

• The NDB_STORED_USER privilege is not supported prior to NDB 8.0.18.

• NDB Cluster 8.0.18 and later supports up to 144 data nodes, while earlier versions support a maximum of only 48 data nodes. See Section 6.23.2.1, "Restoring to Fewer Nodes Than the Original", for information with situations in which this incompatibility causes an issue.

6.23.1.2 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 8.0.14 use .frm files for their metadata. These files can be read by the mysqld in NDB 8.0.14 and later, 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 mysql dump 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 Preparing Your Installation for Upgrade, for more information.

You also 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.

6.23.2 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.

6.23.2.1 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:

```ini
[ndbd]
NodeId=2
HostName=host2
[ndbd]
NodeId=4
HostName=host4
[ndbd]
```
Restoring to a different number of data nodes

We assume that each data node was originally started with `ndbmd --ndb-connectstring=host10` or the equivalent.

2. Perform a backup in the normal manner. See Section 7.8.2, “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-\( B \).\( N \).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 `ndbmd -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
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:

```shell
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:

```shell
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
```

### 6.23.2.2 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:

   ```shell
   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.

   ```shell
   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
Restoring from a backup taken in parallel

- **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:

       ```shell
       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:

       ```shell
       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:

     ```shell
     ndb_restore --nodeid=1 --backupid=1 --restore-data --backup-path=/backups/node_1/BACKUP/BACKUP-1
     ndb_restore --nodeid=2 --backupid=1 --restore-data --backup-path=/backups/node_2/BACKUP/BACKUP-1
     ndb_restore --nodeid=3 --backupid=1 --restore-data --backup-path=/backups/node_3/BACKUP/BACKUP-1
     ndb_restore --nodeid=4 --backupid=1 --restore-data --backup-path=/backups/node_4/BACKUP/BACKUP-1
     ```

     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:

     ```shell
     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.

**6.23.3 Restoring from a backup taken in parallel**

Beginning with NDB Cluster 8.0.16, it is possible to take parallel backups on each data node using `ndbmtd` with multiple LDMs (see Section 7.8.5, “Taking an NDB Backup with Parallel Data Nodes”). The next two sections describe how to restore backups that were taken in this fashion.

**6.23.3.1 Restoring a parallel backup in parallel**

Restoring a parallel backup in parallel requires an `ndb_restore` binary from an NDB Cluster distribution version 8.0.16 or later. 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:

```shell
shell> ndb_restore -n 1 -b 1 -m --backup-path=path/to/backup_dir/BACKUP/BACKUP-backup_id
shell> 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).

6.23.3.2 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:

```shell
shell> 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:

```shell
shell> ndb_restore -n 1 -b 1 -r --backup-path=/opt/BACKUP/BACKUP-100/BACKUP-100-PART-1-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup-path=/opt/BACKUP/BACKUP-100/BACKUP-100-PART-2-OF-4
shell> ndb_restore -n 1 -b 1 -r --backup-path=/opt/BACKUP/BACKUP-100/BACKUP-100-PART-3-OF-4
shell> 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 (prior to NDB 8.0.16) that does not support parallel backups, using the `ndb_restore` binary supplied with the older version of the NDB Cluster software.

6.24 `ndb_select_all` — Print Rows from an NDB Table

`ndb_select_all` prints all rows from an NDB table to stdout.

**Usage**

```bash
ndb_select_all -c connection_string tbl_name -d db_name [ > file_name]
```

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_select_all`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_select_all`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--database=dbname</code>, <code>-d</code></td>
<td>Name of database in which table is found</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--parallelism=#</code>, <code>-P</code></td>
<td>Degree of parallelism</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--lock=#</code>, <code>-l</code></td>
<td>Lock type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--order=index, -o</td>
<td>Sort resultset according to index having this name</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td></td>
<td>Sort resultset in descending order (requires --order)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-z --header, -h</td>
<td>Print header (set to 0</td>
<td>FALSE to disable headers in output)</td>
</tr>
<tr>
<td></td>
<td>Output numbers in hexadecimal format</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--delimiter=char, -D</td>
<td>Set column delimiter</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--disk</td>
<td>Print disk references (useful only for Disk Data tables having nonindexed columns)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--rowid</td>
<td>Print row ID</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--gci</td>
<td>Include GCI in output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--gci64</td>
<td>Include GCI and row epoch in output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--tupscan, -t</td>
<td>Scan in tup order</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nodata</td>
<td>Do not print table column data</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

- **--database=dbname, -d dbname**
  Name of the database in which the table is found. The default value is TEST_DB.

- **parallelism=#, -p #**
  Specifies the degree of parallelism.

- **--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.
• **--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.

• **--descending, -z**

Sorts the output in descending order. This option can be used only in conjunction with the *-o (--order)* option.

• **--header=FALSE**

Excludes column headers from the output.

• **--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.

• **--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.

• **--rowid**

Adds a ROWID column providing information about the fragments in which rows are stored.

• **--gci**

Adds a GCI column to the output showing the global checkpoint at which each row was last updated. See Chapter 3, *NDB Cluster Overview*, and Section 7.3.2, “NDB Cluster Log Events”, for more information about checkpoints.

• **--gci64**

Adds a ROWSGCI64 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.

• **--tupscan, -t**

Scan the table in the order of the tuples.

• **--nodata**

Causes any table data to be omitted.

**Sample Output**

Output from a MySQL *SELECT* statement:

```sql
mysql> SELECT * FROM ctest1.fish;
```

352
ndb_select_count — Print Row Counts for NDB Tables

<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
<td>shark</td>
</tr>
<tr>
<td>6</td>
<td>puffer</td>
</tr>
<tr>
<td>2</td>
<td>tuna</td>
</tr>
<tr>
<td>4</td>
<td>manta ray</td>
</tr>
<tr>
<td>5</td>
<td>grouper</td>
</tr>
<tr>
<td>1</td>
<td>guppy</td>
</tr>
</tbody>
</table>

6 rows in set (0.04 sec)

Output from the equivalent invocation of ndb_select_all:

shell> ./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:

shell> ./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

6.25 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] -d db_name tbl_name[, tbl_name2[, ...]]

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_select_count. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_select_count), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

---

6.25 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] -d db_name tbl_name[, tbl_name2[, ...]]

The following table includes options that are specific to the NDB Cluster native backup restoration program ndb_select_count. Additional descriptions follow the table. For options common to most NDB Cluster programs (including ndb_select_count), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

---

353
### Table 6.22 Command-line options used with the program ndb_select_count

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--database=dbname,</td>
<td>Name of database in which table</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-d</td>
<td>is found</td>
<td></td>
</tr>
<tr>
<td>--parallelism=#,</td>
<td>Degree of parallelism</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-p</td>
<td></td>
<td></td>
</tr>
<tr>
<td>--lock=#,</td>
<td>Lock type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>-l</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

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

```shell
shell> ./ndb_select_count -c localhost -d ctest1 fish dogs
6 records in table fish
4 records in table dogs
NDBT_ProgramExit: 0 - OK
```

### 6.26 ndb_setup.py — Start browser-based Auto-Installer for NDB Cluster (DEPRECATED)

**Note**

This feature is deprecated and should be avoided. It is subject to removal in a future version of NDB Cluster.

**ndb_setup.py** starts the NDB Cluster Auto-Installer and opens the installer’s Start page in the default Web browser.

**Important**

This program is intended to be invoked as a normal user, and not with the `mysql`, `system root` or other administrative account.

This section describes usage of and program options for the command-line tool only. For information about using the Auto-Installer GUI that is spawned when **ndb_setup.py** is invoked, see The NDB Cluster Auto-Installer (NDB 7.5) (No longer supported).

### Usage

**All platforms:**

```bash
ndb_setup.py [options]
```

Additionally, on Windows platforms only:

```bash
setup.bat [options]
```

The following table includes all options that are supported by the NDB Cluster installation and configuration program **ndb_setup.py**. Additional descriptions follow the table.

...
#### Table 6.23 Command-line options used with the program ndb_setup.py

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--browser-start-page=filename, -s</td>
<td>Page that web browser opens when starting</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--ca-certs-file=filename, -a</td>
<td>File containing list of client certificates allowed to connect to server</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--cert-file=filename, -c</td>
<td>File containing X509 certificate identifying server</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--debug-level=level, -d</td>
<td>Python logging module debug level; one of DEBUG, INFO, WARNING (default), ERROR, or CRITICAL</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--help, -h</td>
<td>Print help message</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--key-file=file, -k</td>
<td>Specify file containing private key (if not included in --cert-file)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--no-browser, -n</td>
<td>Do not open start page in browser, merely start tool</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--port=#, -p</td>
<td>Specify port used by web server</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--server-log-file=file, -o</td>
<td>Log requests to this file; use '-' to force logging to stderr instead</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--server-name=name, -N</td>
<td>Name of server to connect to</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--use-http, -H</td>
<td>Use unencrypted (HTTP) client/server connection</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--use-https, -S</td>
<td>Use encrypted (HTTPS) client/server connection</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

- **--browser-start-page=filename, -s**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--browser-start-page=filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>index.html</td>
</tr>
</tbody>
</table>

Specify the file to open in the browser as the installation and configuration Start page. The default is index.html.

- **--ca-certs-file=filename, -a**
Usage

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--ca-certs-file=filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Specify a file containing a list of client certificates which are allowed to connect to the server. The default is an empty string, which means that no client authentication is used.

•  
  \( \text{--cert-file=\textit{file}, -c} \)

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--cert-file=filename</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>/usr/share/mysql/mcc/cfg.pem</td>
</tr>
</tbody>
</table>

Specify a file containing an X509 certificate which identifies the server. It is possible for the certificate to be self-signed. The default is \textit{cfg.pem}.

•  
  \( \text{--debug-level=\textit{level}, -d} \)

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--debug-level=level</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Enumeration</td>
</tr>
<tr>
<td>Default Value</td>
<td>WARNING</td>
</tr>
<tr>
<td>Valid Values</td>
<td>WARNING, DEBUG, INFO, ERROR, CRITICAL</td>
</tr>
</tbody>
</table>

Set the Python logging module debug level. This is one of \texttt{DEBUG, INFO, WARNING, ERROR, or CRITICAL}. \texttt{WARNING} is the default.

•  
  \( \text{--help, -h} \)

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--help</th>
</tr>
</thead>
</table>

Print a help message.

•  
  \( \text{--key-file=\textit{file}, -d} \)

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--key-file=file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Specify a file containing the private key if this is not included in the X509 certificate file (\texttt{--cert-file}). The default is an empty string, which means that no such file is used.

•  
  \( \text{--no-browser, -n} \)

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--no-browser</th>
</tr>
</thead>
</table>

Start the installation and configuration tool, but do not open the Start page in a browser.

•  
  \( \text{--port=\#, -p} \)
**ndb_show_tables** — Display List of NDB Tables

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--port=#</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Numeric</td>
</tr>
<tr>
<td>Default Value</td>
<td>8081</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>65535</td>
</tr>
</tbody>
</table>

Set the port used by the web server. The default is 8081.

- **--server-log-file=file, -o**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--server-log-file=file</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>File name</td>
</tr>
<tr>
<td>Default Value</td>
<td>ndb_setup.log</td>
</tr>
<tr>
<td>Valid Values</td>
<td>ndb_setup.log</td>
</tr>
</tbody>
</table>

Log requests to this file. The default is `ndb_setup.log`. To specify logging to `stderr`, rather than to a file, use a `--` (dash character) for the file name.

- **--server-name=host, -N**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--server-name=name</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>String</td>
</tr>
<tr>
<td>Default Value</td>
<td>localhost</td>
</tr>
</tbody>
</table>

Specify the host name or IP address for the browser to use when connecting. The default is `localhost`.

- **--use-http, -H**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--use-http</th>
</tr>
</thead>
</table>

Make the browser use HTTP to connect with the server. This means that the connection is unencrypted and not secured in any way.

- **--use-https, -S**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--use-https</th>
</tr>
</thead>
</table>

Make the browser use a secure (HTTPS) connection with the server.

6.27 **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.

The following table includes options that are specific to the NDB Cluster native backup restoration program `ndb_show_tables`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_show_tables`), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

---

357
Table 6.24 Command-line options used with the program ndb_show_tables

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--database=string, -d</td>
<td>Specifies database in which table is found; database name must be followed by table name</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--loops=#, -l</td>
<td>Number of times to repeat output</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--parsable, -p</td>
<td>Return output suitable for MySQL LOAD DATA statement</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--show-temp-status</td>
<td>Show table temporary flag</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--type=#, -t</td>
<td>Limit output to objects of this type</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--unqualified, -u</td>
<td>Do not qualify table names</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

Usage

`ndb_show_tables [-c connection_string]`

- **--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.

- **--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.

- **--parsable, -p**
  Using this option causes the output to be in a format suitable for use with LOAD DATA.

- **--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**

358
If specified, this causes unqualified object names to be displayed.

**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 6.24, "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.

### 6.28 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`.

The following table includes options that are specific to the NDB Cluster program `ndb_size.pl`. Additional descriptions follow the table. For options common to most NDB Cluster programs (including `ndb_size.pl`), see Section 6.32, "Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs".

**Table 6.25 Command-line options used with the program ndb_size.pl**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--database=dbname</code></td>
<td>Database or databases to examine; a comma-delimited list; default is ALL (use all databases found on server)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--hostname[:port]</code></td>
<td>Specify host and optional port as <code>host[:port]</code></td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--socket=file_name</code></td>
<td>Specify socket to connect to</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--user=string</code></td>
<td>Specify MySQL user name</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--password=string</td>
<td>Specify MySQL user password</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--format=string</td>
<td>Set output format (text or HTML)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--excludetables=tbl_list</td>
<td>Skip any tables in comma-separated list</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--excludedbs=db_list</td>
<td>Skip any databases in comma-separated list</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--savequeries=file</td>
<td>Saves all queries on database into file specified</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--loadqueries=file</td>
<td>Loads all queries from file specified; does not connect to database</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--real_table_name=table</td>
<td>Designates table to handle unique index size calculations</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

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:

shell> 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):
<table>
<thead>
<tr>
<th>Column Name</th>
<th>Type</th>
<th>Varsized</th>
<th>Key</th>
<th>4.1</th>
<th>5.0</th>
<th>5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>HIDDEN_NDB_PKEY</td>
<td>bigint</td>
<td>PRI</td>
<td>8</td>
<td>8</td>
<td>8</td>
<td></td>
</tr>
<tr>
<td>c2</td>
<td>varchar(50)</td>
<td>Y</td>
<td>52</td>
<td>52</td>
<td>4*</td>
<td></td>
</tr>
<tr>
<td>c1</td>
<td>int(11)</td>
<td></td>
<td>4</td>
<td>4</td>
<td>4</td>
<td></td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Fixed Size Columns DM/Row
|             |          |      | 64 | 64 | 12 |
Varsize Columns DM/Row
|             |          |      | 0  | 0  | 4  |
DataMemory for Indexes:
<table>
<thead>
<tr>
<th>Index Name</th>
<th>Type</th>
<th>4.1</th>
<th>5.0</th>
<th>5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY</td>
<td>BTREE</td>
<td>16</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Total Index DM/Row
|             |          |    | 16  | 16  | 16  |
IndexMemory for Indexes:
<table>
<thead>
<tr>
<th>Index Name</th>
<th>4.1</th>
<th>5.0</th>
<th>5.1</th>
</tr>
</thead>
<tbody>
<tr>
<td>PRIMARY</td>
<td>33</td>
<td>16</td>
<td>16</td>
</tr>
</tbody>
</table>
nbd_top — View CPU usage information for NDB threads

nbd_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.

nbd_top is available beginning with MySQL NDB Cluster 7.6.3.

Usage

nbd_top [-h hostname] [-t port] [-u user] [-p pass] [-n node_id]
**Usage**

`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` and `--port`. To specify a MySQL user and password, use the `--user` and `--passwd` 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 Access Control and Account Management.

Output is available as plain text or an ASCII graph; you can specify this using the `--text` and `--graph` 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`) 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` and `--os-load` 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` 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`.

| 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 6.26 Command-line options used with the program `ndb_top`**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--color</code>, <code>-c</code></td>
<td>Show ASCII graphs in color; use <code>--skip-colors</code> to disable</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--graph</code>, <code>-g</code></td>
<td>Display data using graphs; use <code>--skip-graphs</code> to disable</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--help</code>, <code>-?</code></td>
<td>Show program usage information</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--host[=name]</code>, <code>-h</code></td>
<td>Host name or IP address of MySQL Server to connect to</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
### Additional Options

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--measured-load, -m</td>
<td>Show measured load by thread</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--node-id[=#], -n</td>
<td>Watch node having this node ID</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--os-load, -o</td>
<td>Show load measured by operating system</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--password[=password], -p</td>
<td>Connect using this password</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--port[=#], -P (&gt;7.6.6)</td>
<td>Port number to use when connecting to MySQL Server</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sleep-time[=seconds], -s</td>
<td>Time to wait between display refreshes, in seconds</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--socket, -S</td>
<td>Socket file to use for connection</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--sort, -r</td>
<td>Sort threads by usage; use --skip-sort to disable</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--text, -t (&gt;7.6.6)</td>
<td>Display data using text</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--user[=name], -u</td>
<td>Connect as this MySQL user</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

#### Additional Options

- **--color, -c**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--color</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

  Show ASCII graphs in color; use **--skip-colors** to disable.

- **--graph, -g**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--graph</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Boolean</td>
</tr>
<tr>
<td>Default Value</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

  Display data using graphs; use **--skip-graphs** to disable. This option or **--text** must be true; both options may be true.

- **--help, -?**

<table>
<thead>
<tr>
<th>Command-Line Format</th>
<th>--help</th>
</tr>
</thead>
</table>

  363
## Additional Options

<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>TRUE</td>
</tr>
</tbody>
</table>

Show program usage information.

- **--host[^name^], -h**

  **Command-Line Format**: `--host[^name^]`

  **Type**: String

  **Default Value**: `localhost`

  Host name or IP address of MySQL Server to connect to.

- **--measured-load, -m**

  **Command-Line Format**: `--measured-load`

  **Type**: Boolean

  **Default Value**: FALSE

  Show measured load by thread. This option or **--os-load** must be true; both options may be true.

- **--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`

  **Type**: Boolean

  **Default Value**: TRUE

  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**: Boolean

  **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`.)

- `--sleep-time[=seconds], -s`

  Command-Line Format | `--sleep-time[=seconds]`
  --------- | ---------------
  Type     | Integer
  Default Value | `1`

  Time to wait between display refreshes, in seconds.

- `--socket=path/to/file, -S`

  Command-Line Format | `--socket`
  --------- | ---------------
  Type     | Path name
  Default Value | `[none]`

  Use the specified socket file for the connection.

- `--sort, -r`

  Command-Line Format | `--sort`
  --------- | ---------------
  Type     | Boolean
  Default Value | `TRUE`

  Sort threads by usage; use `--skip-sort` to disable.

- `--text, -t`

  Command-Line Format | `--text`
  --------- | ---------------
  Type     | Boolean
  Default Value | `FALSE`

  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.)

- `--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 `ndbmt` data node under a moderate load. Here, the program has been invoked using `/usr/local/bin/ndb_top -n8 -x` to provide both text and graph output:
Beginning with NDB 8.0.20, ndb_top also shows spin times for threads, displayed in green.

6.30 **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.

The following table includes options that are specific to the NDB Cluster native backup restoration program *ndb_waiter*. Additional descriptions follow the table. For options common to most NDB Cluster programs (including *ndb_waiter*), see Section 6.32, “Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs”.

### Table 6.27 Command-line options used with the program *ndb_waiter*

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--no-contact</code>, <code>-n</code></td>
<td>Wait for cluster to reach NO CONTACT state</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>
Usage

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td>--not-started</td>
<td>Wait for cluster to reach NOT STARTED state</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--single-user</td>
<td>Wait for cluster to enter single user mode</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--timeout=#, -t</td>
<td>Wait this many seconds, then exit whether or not cluster has reached desired state</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--nowait-nodes=list</td>
<td>List of nodes not to be waited for</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--wait-nodes=list, -w</td>
<td>List of nodes to be waited for</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

Usage

ndb_waiter [-c connection_string]

Additional Options

• **--no-contact, -n**
  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NO_CONTACT status before exiting.

• **--not-started**
  Instead of waiting for the STARTED state, ndb_waiter continues running until the cluster reaches NOT_STARTED status before exiting.

• **--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.

• **--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:

  shell> ndb_waiter --nowait-nodes=1,3,7-9

  Important
  Do not use this option together with the --wait-nodes option.

• **--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:

  shell> 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.

```
shell> ./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 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.

### 6.31 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 6.28 Command-line options used with the program ndbxfrm

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--compress</code>, <code>-c</code></td>
<td>Compress file</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--decrypt-password=string</code></td>
<td>Use this password to decrypt file</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--decrypt-password-from-stdin[value]</code></td>
<td>Get decryption password in a secure fashion from STDIN</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td><code>--encrypt-kdf-iter-count=#</code>, <code>-k</code></td>
<td>Number of iterations used in key definition</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--encrypt-password=string</code></td>
<td>Use this password to encrypt file</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--encrypt-password-from-stdin[value]</code></td>
<td>Get encryption password in a secure fashion from STDIN</td>
<td>ADDED: NDB 8.0.24</td>
</tr>
<tr>
<td><code>--help</code>, <code>-?</code></td>
<td>Print usage information</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--info</code>, <code>-i</code></td>
<td>Print file information</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--usage</code>, <code>-?</code></td>
<td>Prints usage information; synonym for --help</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
<tr>
<td><code>--version</code>, <code>-V</code></td>
<td>Output version information</td>
<td>ADDED: NDB 8.0.22</td>
</tr>
</tbody>
</table>

**Usage**

```bash
ndbxfrm --info file[ file ...]
ndbxfrm --compress input_file output_file
ndbxfrm --decrypt-password=password input_file output_file
ndbxfrm [--encrypt-kdf-iter-count=#] --decrypt-password=password input_file output_file
```

*`input_file` and `output_file` cannot be the same file.*

**Options**

- **`--compress`, `-c`**

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-password=password`**

Decrypts a file encrypted by NDB using the password supplied.

- **`--decrypt-password-from-stdin[=TRUE|FALSE]`**

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.
Options

- **--encrypt-kdf-iter-count=#**, -k #
  
  When encrypting a file, specifies the number of iterations to use for the encryption key. Requires the **--encrypt-password** option.

- **--encrypt-password=password**
  
  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

  It is possible but not recommended to use an empty password ("" or "").

- **--encrypt-password-from-stdin[=TRUE|FALSE]**
  
  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, -?**
  
  Prints usage information for the program.

- **--info, -i**
  
  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:

  ```bash
  shell> 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
  ```

- **--usage, -?**
  
  Synonym for **--help**.

- **--version, -V**
  
  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:

```bash
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
All NDB Cluster programs accept the options described in this section, with the following exceptions:

- `mysqld`
- `ndb_print_backup_file`
- `ndb_print_schema_file`
- `ndb_print_sys_file`

**Note**

Users of earlier NDB Cluster versions should note that some of these options have been changed to make them consistent with one another, and also with `mysqld`. You can use the `--help` option with any NDB Cluster program—with the exception of `ndb_print_backup_file`, `ndb_print_schema_file`, and `ndb_print_sys_file`—to view a list of the options which the program supports.

The options in the following table are common to all NDB Cluster executables (except those noted previously in this section).

**Table 6.29 Command-line options common to all MySQL NDB Cluster programs**

<table>
<thead>
<tr>
<th>Format</th>
<th>Description</th>
<th>Added, Deprecated, or Removed</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>--character-sets-dir=dir_name</code></td>
<td>Directory where character sets are installed</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connect-retries=#</code></td>
<td>Set the number of times to retry a connection before giving up</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--connect-retry-delay=#</code></td>
<td>Time to wait between attempts to contact a management server, in seconds</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--core-file</code></td>
<td>Write core on errors (defaults to TRUE in debug builds)</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--debug=options</code></td>
<td>Enable output from debug calls. Can be used only for versions compiled with debugging enabled</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--defaults-extra-file=filename</code></td>
<td>Read this file after global option files are read</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--defaults-file=filename</code></td>
<td>Read default options from this file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--defaults-group-suffix</code></td>
<td>Also read groups with names ending in this suffix</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td><code>--help,</code></td>
<td>Display help message and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>Format</td>
<td>Description</td>
<td>Added, Deprecated, or Removed</td>
</tr>
<tr>
<td>---------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------------------------------------</td>
</tr>
<tr>
<td>--usage, -?</td>
<td></td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--login-path=path</td>
<td>Read this path from the login file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--ndb-connectstring=connectstring</td>
<td>Set connection string for connecting to ndb_mgmd. Syntax: [nodeid=&lt;id&gt;]; [host=&lt;hostname&gt;]:&lt;port&gt;. Overrides entries specified in NDB_CONNECTSTRING or my.cnf</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--connectstring=connectstring, -c</td>
<td></td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--ndb-mgmd-host=host[:port]</td>
<td>Set the host (and port, if desired) for connecting to management server</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--ndb-nodeid=#</td>
<td>Set node id for this node</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--ndb-optimized-node-selection</td>
<td>Select nodes for transactions in a more optimal way</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--no-defaults</td>
<td>Do not read default options from any option file other than login file</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--print-defaults</td>
<td>Print the program argument list and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
<tr>
<td>--version, -V</td>
<td>Output version information and exit</td>
<td>(Supported in all NDB releases based on MySQL 8.0)</td>
</tr>
</tbody>
</table>

For options specific to individual NDB Cluster programs, see Chapter 6, *NDB Cluster Programs*.

See Section 5.3.9.1, “MySQL Server Options for NDB Cluster”, for mysql options relating to NDB Cluster.

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

  Command-Line Format  
  --character-sets-dir=dir_name

  Type  
  Directory name

  Default Value

  Tells the program where to find character set information.

- **--connect-retries=#**

  Command-Line Format  
  --connect-retries=#

  Type  
  Numeric

  Default Value  
  12

  Minimum Value  
  0

  Maximum Value  
  4294967295

  This option specifies the number of times following the first attempt to retry a connection before giving up (the client always tries the connection at least once). The length of time to wait per attempt is set using **--connect-retry-delay**.
Options Common to NDB Cluster Programs — Options Common to NDB Cluster Programs

Note
When used with `ndb_mgm`, this option has 3 as its default. See Section 6.5, "ndb_mgm — The NDB Cluster Management Client", for more information.

- `--connect-retry-delay=`

  Command-Line Format: `--connect-retry-delay=`
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Numeric</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>5</td>
</tr>
<tr>
<td>Minimum Value</td>
<td>1</td>
</tr>
<tr>
<td>Maximum Value</td>
<td>4294967295</td>
</tr>
</tbody>
</table>

  This option specifies the length of time to wait per attempt a connection before giving up. The number of times to try connecting is set by `--connect-retries`.

- `--core-file`

  Command-Line Format: `--core-file`
  
<table>
<thead>
<tr>
<th>Type</th>
<th>Boolean</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>FALSE</td>
</tr>
</tbody>
</table>

  Write a core file if the program dies. The name and location of the core file are system-dependent. (For NDB Cluster programs nodes running on Linux, the default location is the program's working directory—for a data node, this is the node's `DataDir`.) For some systems, there may be restrictions or limitations. For example, it might be necessary to execute `ulimit -c unlimited` before starting the server. Consult your system documentation for detailed information.

  If NDB Cluster was built using the `--debug` option for `configure`, then `--core-file` is enabled by default. For regular builds, `--core-file` is disabled by default.

- `--debug[=options]`

  Command-Line Format: `--debug=options`
  
<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td><code>d:t:O,/tmp/ndb_restore.trace</code></td>
</tr>
</tbody>
</table>

  This option can be used only for versions compiled with debugging enabled. It is used to enable output from debug calls in the same manner as for the `mysqld` process.

- `--defaults-extra-file=filename`

  Command-Line Format: `--defaults-extra-file=filename`
  
<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

  Read this file after global option files are read.

  For additional information about this and other option-file options, see Command-Line Options that Affect Option-File Handling.
• **--defaults-file=filename**

Command-Line Format: `--defaults-file=filename`

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Read default options from this file.

For additional information about this and other option-file options, see [Command-Line Options that Affect Option-File Handling](#).

• **--defaults-group-suffix**

Command-Line Format: `--defaults-group-suffix`

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Also read groups with names ending in this suffix.

For additional information about this and other option-file options, see [Command-Line Options that Affect Option-File Handling](#).

• **--help, --usage, -?**

Command-Line Format: `--help`, `--usage`

Prints a short list with descriptions of the available command options.

• **--login-path=path**

Command-Line Format: `--login-path=path`

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>[none]</td>
</tr>
</tbody>
</table>

Read this path from the login file.

For additional information about this and other option-file options, see [Command-Line Options that Affect Option-File Handling](#).

• **--ndb-connectstring=connection_string, --connect-string=connection_string, -c connection_string**

Command-Line Format: `--ndb-connectstring=connectstring`, `--connect-string=connectstring`

<table>
<thead>
<tr>
<th>Type</th>
<th>String</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default Value</td>
<td>localhost:1186</td>
</tr>
</tbody>
</table>

This option takes an NDB Cluster connection string that specifies the management server for the application to connect to, as shown here:

```sh
cshell> ndbd --ndb-connectstring="nodeid=2;host=ndb_mgmd.mysql.com:1186"
```

For more information, see [Section 5.3.3, “NDB Cluster Connection Strings”](#).
• \texttt{--ndb-mgmd-host=host[:port]}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--ndb-mgmd-host=host[:port]}

\textbf{Type} \hspace{1cm} String

\textbf{Default Value} \hspace{1cm} localhost:1186

Can be used to set the host and port number of a single management server for the program to connect to. If the program requires node IDs or references to multiple management servers (or both) in its connection information, use the \texttt{--ndb-connectstring} option instead.

• \texttt{--ndb-nodeid=#}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--ndb-nodeid=#}

\textbf{Type} \hspace{1cm} Numeric

\textbf{Default Value} \hspace{1cm} 0

Sets this node's NDB Cluster node ID. The range of permitted values depends on the node's type (data, management, or API) and the NDB Cluster software version. See Section 3.7.2, "Limits and Differences of NDB Cluster from Standard MySQL Limits", for more information.

• \texttt{--no-defaults}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--no-defaults}

\textbf{Type} \hspace{1cm} Boolean

\textbf{Default Value} \hspace{1cm} TRUE

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

For additional information about this and other option-file options, see Command-Line Options that Affect Option-File Handling.

• \texttt{--ndb-optimized-node-selection}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--ndb-optimized-node-selection}

\textbf{Type} \hspace{1cm} Boolean

\textbf{Default Value} \hspace{1cm} TRUE

Optimize selection of nodes for transactions. Enabled by default.

• \texttt{--print-defaults}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--print-defaults}

\textbf{Type} \hspace{1cm} Boolean

\textbf{Default Value} \hspace{1cm} TRUE

Print the program argument list and exit.

For additional information about this and other option-file options, see Command-Line Options that Affect Option-File Handling.

• \texttt{--version}, \texttt{-V}

\textbf{Command-Line Format} \hspace{1cm} \texttt{--version}

Prints the NDB Cluster version number of the executable. The version number is relevant because not all versions can be used together, and the NDB Cluster startup process verifies that the versions
of the binaries being used can co-exist in the same cluster. This is also important when performing an online (rolling) software upgrade or downgrade of NDB Cluster.

See Section 7.5, “Performing a Rolling Restart of an NDB Cluster”), for more information.
Chapter 7 Management of NDB Cluster

Table of Contents

7.1 Commands in the NDB Cluster Management Client .................................................. 379
7.2 NDB Cluster Log Messages ......................................................................................... 384
  7.2.1 NDB Cluster: Messages in the Cluster Log ......................................................... 384
  7.2.2 NDB Cluster Log Startup Messages ................................................................. 396
  7.2.3 Event Buffer Reporting in the Cluster Log ......................................................... 396
  7.2.4 NDB Cluster: NDB Transporter Errors ............................................................. 397
7.3 Event Reports Generated in NDB Cluster ............................................................... 399
  7.3.1 NDB Cluster Logging Management Commands .............................................. 400
  7.3.2 NDB Cluster Log Events ................................................................................. 402
  7.3.3 Using CLUSTERLOG STATISTICS in the NDB Cluster Management Client ...... 407
7.4 Summary of NDB Cluster Start Phases ................................................................. 409
7.5 Performing a Rolling Restart of an NDB Cluster .................................................... 411
7.6 NDB Cluster Single User Mode .............................................................................. 413
7.7 Adding NDB Cluster Data Nodes Online .............................................................. 414
  7.7.1 Adding NDB Cluster Data Nodes Online: General Issues ................................ 414
  7.7.2 Adding NDB Cluster Data Nodes Online: Basic procedure ............................. 416
  7.7.3 Adding NDB Cluster Data Nodes Online: Detailed Example ......................... 417
7.8 Online Backup of NDB Cluster .............................................................................. 424
  7.8.1 NDB Cluster Backup Concepts ...................................................................... 424
  7.8.2 Using The NDB Cluster Management Client to Create a Backup ..................... 425
  7.8.3 Configuration for NDB Cluster Backups ............................................................ 428
  7.8.4 NDB Cluster Backup Troubleshooting ............................................................... 429
  7.8.5 Taking an NDB Backup with Parallel Data Nodes ............................................ 429
7.9 MySQL Server Usage for NDB Cluster ............................................................... 430
7.10 NDB Cluster Disk Data Tables .............................................................................. 431
  7.10.1 NDB Cluster Disk Data Objects ..................................................................... 431
  7.10.2 NDB Cluster Disk Data Storage Requirements ............................................. 436
7.11 Online Operations with ALTER TABLE in NDB Cluster ....................................... 437
7.12 Distributed MySQL Privileges with NDB_STORED_USER .................................... 440
7.13 NDB API Statistics Counters and Variables .......................................................... 441
7.14 ndbinfo: The NDB Cluster Information Database .................................................. 451
  7.14.1 The ndbinfo arbitrator_validity_detail Table .................................................. 455
  7.14.2 The ndbinfo arbitrator_validity_summary Table .......................................... 456
  7.14.3 The ndbinfo backup_id Table ....................................................................... 456
  7.14.4 The ndbinfo blocks Table ............................................................................ 457
  7.14.5 The ndbinfo cluster_locks Table .................................................................... 458
  7.14.6 The ndbinfo cluster_operations Table ......................................................... 459
  7.14.7 The ndbinfo cluster_transactions Table ...................................................... 460
  7.14.8 The ndbinfo config_nodes Table ................................................................. 462
  7.14.9 The ndbinfo config_params Table ................................................................ 462
  7.14.10 The ndbinfo config_values Table ................................................................. 463
  7.14.11 The ndbinfo counters Table ....................................................................... 465
  7.14.12 The ndbinfo cpudata Table ....................................................................... 467
  7.14.13 The ndbinfo cpudata_1sec Table .................................................................. 468
  7.14.14 The ndbinfo cpudata_20sec Table ................................................................. 468
  7.14.15 The ndbinfo cpudata_50ms Table ................................................................. 469
  7.14.16 The ndbinfo cpuinto Table ....................................................................... 470
  7.14.17 The ndbinfo cpustat Table ....................................................................... 471
  7.14.18 The ndbinfo cpustat_50ms Table ................................................................. 471
  7.14.19 The ndbinfo cpustat_1sec Table .................................................................. 472
  7.14.20 The ndbinfo cpustat_20sec Table ................................................................. 473
  7.14.21 The ndbinfo dict_obj_info Table ................................................................. 474
Managing an NDB Cluster involves a number of tasks, the first of which is to configure and start NDB Cluster. This is covered in Chapter 5, Configuration of NDB Cluster, and Chapter 6, 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 7.17, “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 7.14, “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.

### 7.1 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 7.3, “Event Reports Generated in NDB Cluster”; commands for creating backups and restoring from them are provided in Section 7.8, “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.

- **HELP**
  Stored information on all available commands.

- **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.

- **SHOW**
Displays information on the cluster's status. 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 (status **SINGLE USER MODE**). In NDB 8.0.17 and later, 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 version 8.0.17 or later.

- **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 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.

- **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.

- **node_id STATUS**
  Displays status information for the data node identified by `node_id` (or for all data nodes).
  The output from this command also indicates when the cluster is in single user mode.

- **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.

- **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.

  Beginning with NDB 8.0.17, 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.
  ``
  Also in NDB 8.0.17 and later, 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-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0, *)
  id=6 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17, single user mode, Nodegroup: 0)
  [ndb_mgm(MGM)] 1 node(s)
  id=50 @127.0.0.1 (mysql-8.0.17 ndb-8.0.17)
  ```
Commands in the NDB Cluster Management Client

[mysqld(API)] 2 node(s)
id=100  @127.0.0.1  (mysql-8.0.17 ndb-8.0.17, allowed single user)
id=101 (not connected, accepting connect from any host)

Note
All data and management nodes must be running version 8.0.17 of the NDB Cluster software for this feature to be enabled.

• 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.

• QUIT, EXIT

Terminates the management client.
This command does not affect any nodes connected to the cluster.

• 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.

• 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 that are to join the new node group. The number of nodes 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 has 2 node groups of 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 7.7, “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  (8.0.24-ndb-8.0.24, no nodegroup)
Commands in the NDB Cluster Management Client

**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 7.7, “Adding NDB Cluster Data Nodes Online”.

- **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
jon@valhaj:~/bin>
```

Note that leading spaces and spaces within the `prompt` string are not trimmed. Trailing spaces are removed.

- **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.

**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 7.8, “Online Backup of NDB Cluster”.

- The **CLUSTERLOG** command is used to perform various logging functions. See Section 7.3, “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 MySQL NDB Cluster Internals Manual.
### 7.2 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.

#### 7.2.1 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 7.3, “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>
<thead>
<tr>
<th>Log Message</th>
<th>Description</th>
<th>Event Name</th>
<th>Event Type</th>
<th>Priority</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node</td>
<td>The data node having node ID node_id has connected to the management server (node mgm_node_id).</td>
<td>Connected</td>
<td>Connection</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td></td>
<td>The data node having node ID data_node_id has disconnected from the management server (node mgm_node_id).</td>
<td>Disconnected</td>
<td>Connection</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>Node</td>
<td>The API node or SQL node having node ID api_node_id is no longer communicating with data node data_node_id.</td>
<td>CommunicationClosed</td>
<td>Connection</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td></td>
<td>The API node or SQL node having node ID api_node_id is now communicating with data node data_node_id.</td>
<td>CommunicationOpened</td>
<td>Connection</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node</td>
<td>The API node having node ID api_node_id has connected to management node mgm_node_id using NDB API version (generally the same as the</td>
<td>ConnectedApiVersion</td>
<td>Connection</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Node node_id: Global checkpoint gci started</td>
<td>A global checkpoint with the ID gci has been started; node node_id is the master responsible for this global checkpoint.</td>
<td>GlobalCheckpointStarted</td>
<td>Checkpoint</td>
<td>9</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Global checkpoint gci completed</td>
<td>The global checkpoint having the ID gci has been completed; node node_id was the master responsible for this global checkpoint.</td>
<td>GlobalCheckpointCompleted</td>
<td>Checkpoint</td>
<td>10</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Local checkpoint lcp started. Keep GCI = current_gci oldest restorable GCI = old_gci</td>
<td>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.</td>
<td>LocalCheckpointStarted</td>
<td>Checkpoint</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Local checkpoint lcp completed</td>
<td>The local checkpoint having sequence ID lcp on node node_id has been completed.</td>
<td>LocalCheckpointCompleted</td>
<td>Checkpoint</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Local Checkpoint stopped in CALCULATED_KEEP_GCI</td>
<td>The node was unable to determine the most recent usable GCI.</td>
<td>LCPStoppedInCalcKeepGci</td>
<td>Checkpoint</td>
<td>0</td>
<td>ALERT</td>
</tr>
<tr>
<td>Node node_id: Table ID = table_id, fragment ID = fragment_id has completed LCP on Node node_id maxGciStarted: started_gci maxGciCompleted: completed_gci</td>
<td>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</td>
<td>LCPFragmentCompleted</td>
<td>Checkpoint</td>
<td>11</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------</td>
<td>--------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Node node_id: ACC Blocked num_1 and TUP Blocked num_2 times last second</td>
<td>Undo logging is blocked because the log buffer is close to overflowing.</td>
<td>UndoLogBlocked</td>
<td>Checkpoint</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Start initiated version</td>
<td>Data node node_id, running NDB version version, is beginning its startup process.</td>
<td>NDBStartStarted</td>
<td>StartUp</td>
<td>1</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Started version</td>
<td>Data node node_id, running NDB version version, has started successfully.</td>
<td>NDBStartCompleted</td>
<td>StartUp</td>
<td>1</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: STTORYY received after restart finished</td>
<td>The node has received a signal indicating that a cluster restart has completed.</td>
<td>STTORYRecieved</td>
<td>StartUp</td>
<td>15</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Start phase phase completed (type)</td>
<td>The node has completed start phase phase of a type start. For a listing of start phases, see Section 7.4, “Summary of NDB Cluster Start Phases”. (type is one of initial, system, node, initial node, or &lt;Unknown&gt;.)</td>
<td>StartPhaseCompleted</td>
<td>StartUp</td>
<td>4</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: CM_REGCONF president = president_id, own Node = own_id, our dynamic id = dynamic_id</td>
<td>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.</td>
<td>CM_REGCONF</td>
<td>StartUp</td>
<td>3</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: CM_REGREF from Node president_id to our Node</td>
<td>The reporting node (ID node_id) was unable to accept node president_id</td>
<td>CM_REGREF</td>
<td>StartUp</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>node_id. Cause = cause</td>
<td>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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node node_id:</td>
<td>The node has discovered its neighboring 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.</td>
<td>FIND_NEIGHBOURS</td>
<td>StartUp</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>type shutdown initiated</td>
<td>The node has received a shutdown signal. The type of shutdown is either Cluster or Node.</td>
<td>NDBStopStarted</td>
<td>StartUp</td>
<td>1</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id:</td>
<td>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.</td>
<td>NDBStopCompleted</td>
<td>StartUp</td>
<td>1</td>
<td>INFO</td>
</tr>
<tr>
<td>Node shutdown completed [, action] [Initiated by signal signal.]</td>
<td>The node has been forcibly shut down. The action (one of restarting, no start, or initial) subsequently being taken, if any, is</td>
<td>NDBStopForced</td>
<td>StartUp</td>
<td>1</td>
<td>ALERT</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>---------------------------------------------------------------------------</td>
<td>-------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>-------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>start_phase.</strong> [Initiated by signal.] [Caused by error error_code: 'error_message' error_status'. [(extra info extra_code)]]**</td>
<td>also reported. If the shutdown occurred while the node was starting, the report includes the error classification 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.</td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td><strong>Node node_id:</strong> Node shutdown aborted</td>
<td>The node shutdown process was aborted by the user.</td>
<td>NDBStopAborted</td>
<td>StartUp</td>
<td>1</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> StartLog: [GCI Keep: keep_pos LastCompleted: last_pos NewestRestorable: restore_pos]**</td>
<td>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 restore all data nodes.</td>
<td>StartREDOLog</td>
<td>StartUp</td>
<td>4</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>startup_message</strong> [Listed separately; see below.]</td>
<td>There are a number of possible startup messages that can be logged under different circumstances. These are listed</td>
<td>StartReport</td>
<td>StartUp</td>
<td>4</td>
<td>INFO</td>
</tr>
</tbody>
</table>
## NDB Cluster: Messages in the Cluster Log

<table>
<thead>
<tr>
<th>Log Message</th>
<th>Description</th>
<th>Event Name</th>
<th>Event Type</th>
<th>Priority</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node node_id: Node restart completed copy of dictionary information</td>
<td>Copying of data dictionary information to the restarted node has been completed.</td>
<td>NR_CopyDict</td>
<td>NodeRestart</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Node restart completed copy of distribution information</td>
<td>Copying of data distribution information to the restarted node has been completed.</td>
<td>NR_CopyDistr</td>
<td>NodeRestart</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Node restart starting to copy the fragments to Node node_id</td>
<td>Copy of fragments to starting data node node_id has begun.</td>
<td>NR_CopyFragsStarted</td>
<td>NodeRestart</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Table ID = table_id, fragment ID = fragment_id have been copied to Node node_id</td>
<td>Fragment fragment_id from table table_id has been copied to data node node_id.</td>
<td>NR_CopyFragDone</td>
<td>NodeRestart</td>
<td>10</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Node restart completed copying the fragments to Node node_id</td>
<td>Copying of all table fragments to restarting data node node_id has been completed.</td>
<td>NR_CopyFragCompleted</td>
<td>NodeRestart</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Node node1_id completed failure of Node node2_id</td>
<td>Data node node1_id has detected the failure of data node node2_id.</td>
<td>NodeFailCompleted</td>
<td>NodeRestart</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>All nodes completed failure of Node node_id</td>
<td>All (remaining) data nodes have detected the failure of data node node_id.</td>
<td>NodeFailCompleted</td>
<td>NodeRestart</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>Node failure of node_idblock completed</td>
<td>The failure of data node node_id has been detected in the blockNDB kernel block, where block is 1 of DBTC, DBDICT, DBDIH, or DBLQH; for more details, see Section 7.2.2, “NDB Cluster Log Startup Messages”.</td>
<td>NodeFailCompleted</td>
<td>NodeRestart</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>----------------------------------------------------------------------------</td>
<td>--------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------</td>
<td>-------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Node mgm_node_id: Node data_node_id has failed. The Node state at failure was state_code</td>
<td>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/signaldta/ArbitSignalData.hpp.</td>
<td>NODE_FAILREP</td>
<td>NodeRestart</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>President restarts arbitration thread node node_id [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]</td>
<td>This is a report on the current 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/signaldta/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.</td>
<td>ArbitState</td>
<td>NodeRestart</td>
<td>6</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Arbitration check lost - less than 1/2 nodes left or Arbitration check won - all node groups and more than 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]</td>
<td>This message reports on the result of arbitration. In the event of arbitration failure, an error_message and an arbitration state_code are provided; definitions for both of these are found in include/kernel/signaldata/ArbitSignalData.hpp.</td>
<td>ArbitResult</td>
<td>NodeRestart</td>
<td>2</td>
<td>ALERT</td>
</tr>
<tr>
<td>Node node_id: GCP Take over started</td>
<td>This node is attempting to assume responsibility for the next global checkpoint (that is, it is becoming the master node)</td>
<td>GCP_TakeoverStarted</td>
<td>NodeRestart</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: GCP Take over completed</td>
<td>This node has become the master, and has assumed responsibility for</td>
<td>GCP_TakeoverCompleted</td>
<td>NodeRestart</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-----------------------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> LCP Takeover started</td>
<td>This node is attempting to assume responsibility for the next set of local checkpoints (that is, it is becoming the master node)</td>
<td>LCP_TakeoverStarted</td>
<td>NodeRestart</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> LCP Takeover completed</td>
<td>This node has become the master, and has assumed responsibility for the next set of local checkpoints</td>
<td>LCP_TakeoverCompleted</td>
<td>NodeRestart</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> 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</td>
<td>This report of transaction activity is given approximately once every 10 seconds</td>
<td>TransReportCounters</td>
<td>Statistic</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> Operations=operations</td>
<td>Number of operations performed by this node, provided approximately once every 10 seconds</td>
<td>OperationReportCounters</td>
<td>Statistic</td>
<td>8</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> Table with ID = table_id created</td>
<td>A table having the table ID shown has been created</td>
<td>TableCreated</td>
<td>Statistic</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td><strong>Node node_id:</strong> Mean loop Counter in doJob last</td>
<td></td>
<td>JobStatistic</td>
<td>Statistic</td>
<td>9</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-----------------------------------------------------------------------------</td>
<td>-----------------------------------------------------------------------------------------------------------------------------------------------</td>
<td>---------------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>8192 times = count</td>
<td>This node is sending an average of bytes per send to node node_id</td>
<td>SendBytesStatistic</td>
<td>Statistic</td>
<td>9</td>
<td>INFO</td>
</tr>
<tr>
<td>Mean send size to Node = node_id last 4096 sends = bytes bytes</td>
<td>This node is receiving an average of bytes of data each time it receives data from node node_id</td>
<td>ReceiveBytesStatistic</td>
<td>Statistic</td>
<td>9</td>
<td>INFO</td>
</tr>
<tr>
<td>Mean receive size to Node = node_id last 4096 sends = bytes bytes</td>
<td>This report is generated when a DUMP 1000 command is issued in the cluster management client</td>
<td>MemoryUsage</td>
<td>Statistic</td>
<td>5</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Data usage is (data_pages_used 32K pages of total data_pages_total) / Node node_id: Index usage is (index_pages_used 8K pages of total index_pages_total)</td>
<td>This report is generated when a DUMP 1000 command is issued in the cluster management client</td>
<td>MemoryUsage</td>
<td>Statistic</td>
<td>5</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node1_id: Transporter to node node2_id reported error error_code: error_message</td>
<td>A transporter error occurred while communicating with node node2_id; for a listing of transporter error codes and messages, see NDB Transporter Errors, in MySQL NDB Cluster Internals Manual</td>
<td>TransporterError</td>
<td>Error</td>
<td>2</td>
<td>ERROR</td>
</tr>
<tr>
<td>Node node1_id: Transporter to node node2_id reported error error_code: error_message</td>
<td>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</td>
<td>TransporterWarning</td>
<td>Error</td>
<td>8</td>
<td>WARNING</td>
</tr>
</tbody>
</table>
### NDB Cluster: Messages in the Cluster Log

<table>
<thead>
<tr>
<th>Log Message</th>
<th>Description</th>
<th>Event Name</th>
<th>Event Type</th>
<th>Priority</th>
<th>Severity</th>
</tr>
</thead>
<tbody>
<tr>
<td>Node node1_id: Node node2_id missed heartbeat heartbeat_id</td>
<td>This node missed a heartbeat from node node2_id</td>
<td>MissedHeartbeat</td>
<td>Error</td>
<td>8</td>
<td>WARNING</td>
</tr>
<tr>
<td>Node node1_id: Node node2_id declared dead due to missed heartbeat</td>
<td>This node has missed at least 3 heartbeats from node node2_id, and so has declared that node “dead”</td>
<td>DeadDueToHeartbeat</td>
<td>Error</td>
<td>8</td>
<td>ALERT</td>
</tr>
<tr>
<td>Node node1_id: Node Sent Heatbeat to node = node2_id</td>
<td>This node has sent a heartbeat to node node2_id</td>
<td>SentHeartbeat</td>
<td>Info</td>
<td>12</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Event buffer status (object_id): used=bytes_used (percent_used% of alloc) alloc=bytes_alloc max=bytes_available latest_consumed_epoch=latest_consumed_epoch latest_buffered_epoch=latest_buffered_epoch report_reason=</td>
<td>This report is seen during heavy event buffer usage, for example, when many 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 7.2.3, “Event Buffer Reporting in the Cluster Log”</td>
<td>EventBufferStatus2</td>
<td>Info</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>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</td>
<td>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 7.6, “NDB SingleUser”</td>
<td>SingleUser</td>
<td>Info</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Log Message</td>
<td>Description</td>
<td>Event Name</td>
<td>Event Type</td>
<td>Priority</td>
<td>Severity</td>
</tr>
<tr>
<td>-------------</td>
<td>-------------</td>
<td>------------</td>
<td>------------</td>
<td>----------</td>
<td>----------</td>
</tr>
<tr>
<td>Cluster Single User Mode&quot;; the message Unknown single user report API_node_id indicates an error has taken place and should never be seen in normal operation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>Node node_id: Backup backup_id started from node mgm_node_id</td>
<td>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 7.8.2, “Using The NDB Cluster Management Client to Create a Backup”</td>
<td>BackupStarted</td>
<td>Backup</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>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</td>
<td>The backup having the ID backup_id has been completed; for more information, see Section 7.8.2, “Using The NDB Cluster Management Client to Create a Backup”</td>
<td>BackupCompleted</td>
<td>Backup</td>
<td>7</td>
<td>INFO</td>
</tr>
<tr>
<td>Node node_id: Backup request from mgm_node_id failed to start</td>
<td>The backup failed to start; for error codes, see MGM API Errors</td>
<td>BackupFailedToStart</td>
<td>Backup</td>
<td>7</td>
<td>ALERT</td>
</tr>
</tbody>
</table>
### 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 start report: 0x%x [ %s %s %s %s ]

### 7.2.3 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:

```plaintext
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 if 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.

### 7.2.4 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.
NDB Cluster: NDB 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
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 5.3.5, “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:

- 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.
NDB Cluster Logging Management Commands

- **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: ALERT, CRITICAL, ERROR, WARNING, INFO, or DEBUG.

Both the cluster log and the node log can be filtered on these properties.

The format used in the cluster log is as shown here:

<table>
<thead>
<tr>
<th>Date</th>
<th>Time</th>
<th>Type</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 1: Data usage is 2% (60 32K pages of total 2560)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 1: Index usage is 1% (24 8K pages of total 2336)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 1: Resource 0 min: 0 max: 639 curr: 0</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 2: Data usage is 2% (76 32K pages of total 2560)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 2: Index usage is 1% (24 8K pages of total 2336)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 2: Resource 0 min: 0 max: 639 curr: 0</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 3: Data usage is 2% (58 32K pages of total 2560)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 3: Index usage is 1% (25 8K pages of total 2336)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 3: Resource 0 min: 0 max: 639 curr: 0</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 4: Data usage is 2% (74 32K pages of total 2560)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 4: Index usage is 1% (25 8K pages of total 2336)</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:35:55</td>
<td>INFO</td>
<td>Node 4: Resource 0 min: 0 max: 639 curr: 0</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:39:42</td>
<td>INFO</td>
<td>Node 9 Connected</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:39:42</td>
<td>INFO</td>
<td>Node 9 Connected</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:39:42</td>
<td>ALERT</td>
<td>Node 7 Disconnected</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:39:42</td>
<td>ALERT</td>
<td>Node 7 Disconnected</td>
</tr>
<tr>
<td>2007-01-26</td>
<td>19:39:42</td>
<td>ALERT</td>
<td>Node 7 Disconnected</td>
</tr>
</tbody>
</table>

Each line in the cluster log contains the following information:

- A timestamp in **YYYY-MM-DD HH:MM:SS** format.
- The type of node which is performing the logging. In the cluster log, this is always [MgmSrvr].
- The severity of the event.
- The ID of the node reporting the event.
- 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 information.

### 7.3.1 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.
NDB Cluster Logging Management Commands

- **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.

<table>
<thead>
<tr>
<th>Category</th>
<th>Default threshold (All data nodes)</th>
</tr>
</thead>
<tbody>
<tr>
<td>STARTUP</td>
<td>7</td>
</tr>
<tr>
<td>SHUTDOWN</td>
<td>7</td>
</tr>
<tr>
<td>STATISTICS</td>
<td>7</td>
</tr>
<tr>
<td>CHECKPOINT</td>
<td>7</td>
</tr>
<tr>
<td>NODERESTART</td>
<td>7</td>
</tr>
<tr>
<td>CONNECTION</td>
<td>7</td>
</tr>
<tr>
<td>ERROR</td>
<td>15</td>
</tr>
<tr>
<td>INFO</td>
<td>7</td>
</tr>
</tbody>
</table>

The **STATISTICS** category can provide a great deal of useful data. See Section 7.3.3, “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.

<table>
<thead>
<tr>
<th>Severity Level Value</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>ALERT</td>
<td>A condition that should be corrected immediately, such as a corrupted system database</td>
</tr>
<tr>
<td>2</td>
<td>CRITICAL</td>
<td>Critical conditions, such as device errors or insufficient resources</td>
</tr>
<tr>
<td>3</td>
<td>ERROR</td>
<td>Conditions that should be corrected, such as configuration errors</td>
</tr>
<tr>
<td>4</td>
<td>WARNING</td>
<td>Conditions that are not errors, but that might require special handling</td>
</tr>
<tr>
<td>5</td>
<td>INFO</td>
<td>Informational messages</td>
</tr>
</tbody>
</table>

**Note**

Events are reported per data node, and that the threshold can be set to different values on different nodes.

**Note**

These correspond to Unix syslog levels, except for LOG_EMERG and LOG_NOTICE, which are not used or mapped.
### NDB Cluster Log Events

<table>
<thead>
<tr>
<th>Severity Level Value</th>
<th>Severity</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>6</td>
<td>DEBUG</td>
<td>Debugging messages used for NDBCLUSTER development</td>
</tr>
</tbody>
</table>

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_mgmd` 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.

### 7.3.2 NDB Cluster Log Events

An event report reported in the event logs has the following format:

```plaintext
datetime [string] severity -- message
```

For example:

```
```

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 7.4 Events associated with connections between cluster nodes**

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>Connected</td>
<td>8</td>
<td>INFO</td>
<td>Data nodes connected</td>
</tr>
<tr>
<td>Disconnected</td>
<td>8</td>
<td>ALERT</td>
<td>Data nodes disconnected</td>
</tr>
<tr>
<td>CommunicationClosed</td>
<td>8</td>
<td>INFO</td>
<td>SQL node or data node connection closed</td>
</tr>
<tr>
<td>CommunicationOpened</td>
<td>8</td>
<td>INFO</td>
<td>SQL node or data node connection open</td>
</tr>
<tr>
<td>ConnectedApiVersion</td>
<td>8</td>
<td>INFO</td>
<td>Connection using API version</td>
</tr>
</tbody>
</table>

### CHECKPOINT Events

The logging messages shown here are associated with checkpoints.

**Table 7.5 Events associated with checkpoints**

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GlobalCheckpointStarted</td>
<td>9</td>
<td>INFO</td>
<td>Start of GCP: REDO log is written to disk</td>
</tr>
<tr>
<td>GlobalCheckpointCompleted</td>
<td>10</td>
<td>INFO</td>
<td>GCP finished</td>
</tr>
<tr>
<td>LocalCheckpointStarted</td>
<td>7</td>
<td>INFO</td>
<td>Start of LCP: data written to disk</td>
</tr>
</tbody>
</table>
### NDB Cluster Log Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>LocalCheckpointCompleted</td>
<td>7</td>
<td>INFO</td>
<td>LCP completed normally</td>
</tr>
<tr>
<td>LCPStoppedInCalcKeepGci</td>
<td>0</td>
<td>ALERT</td>
<td>LCP stopped</td>
</tr>
<tr>
<td>LCPFragmentCompleted</td>
<td>11</td>
<td>INFO</td>
<td>LCP on a fragment has been completed</td>
</tr>
<tr>
<td>UndoLogBlocked</td>
<td>7</td>
<td>INFO</td>
<td>UNDO logging blocked; buffer near overflow</td>
</tr>
<tr>
<td>RedoStatus</td>
<td>7</td>
<td>INFO</td>
<td>Redo status</td>
</tr>
</tbody>
</table>

### 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 7.6 Events relating to the startup of a node or cluster**

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDBStartStarted</td>
<td>1</td>
<td>INFO</td>
<td>Data node start phases initiated (all nodes starting)</td>
</tr>
<tr>
<td>NDBStartCompleted</td>
<td>1</td>
<td>INFO</td>
<td>Start phases completed, all data nodes</td>
</tr>
<tr>
<td>STTorryRecieved</td>
<td>15</td>
<td>INFO</td>
<td>Blocks received after completion of restart</td>
</tr>
<tr>
<td>StartPhaseCompleted</td>
<td>4</td>
<td>INFO</td>
<td>Data node start phase X completed</td>
</tr>
<tr>
<td>CM_REGCONF</td>
<td>3</td>
<td>INFO</td>
<td>Node has been successfully included into the cluster; shows node, managing node, and dynamic ID</td>
</tr>
<tr>
<td>CM_REGREF</td>
<td>8</td>
<td>INFO</td>
<td>Node has been refused for inclusion in the cluster; cannot be included in cluster due to misconfiguration, inability to establish communication, or other problem</td>
</tr>
<tr>
<td>FIND_NEIGHBOURS</td>
<td>8</td>
<td>INFO</td>
<td>Shows neighboring data nodes</td>
</tr>
<tr>
<td>NDBStopStarted</td>
<td>1</td>
<td>INFO</td>
<td>Data node shutdown initiated</td>
</tr>
<tr>
<td>NDBStopCompleted</td>
<td>1</td>
<td>INFO</td>
<td>Data node shutdown complete</td>
</tr>
<tr>
<td>NDBStopForced</td>
<td>1</td>
<td>ALERT</td>
<td>Forced shutdown of data node</td>
</tr>
<tr>
<td>NDBStopAborted</td>
<td>1</td>
<td>INFO</td>
<td>Unable to shut down data node normally</td>
</tr>
<tr>
<td>StartREDOLog</td>
<td>4</td>
<td>INFO</td>
<td>New redo log started; GCI keep X, newest restorable GCI Y</td>
</tr>
<tr>
<td>StartLog</td>
<td>10</td>
<td>INFO</td>
<td>New log started; log part X, start MB Y, stop MB Z</td>
</tr>
<tr>
<td>UNDORecordsExecuted</td>
<td>15</td>
<td>INFO</td>
<td>Undo records executed</td>
</tr>
<tr>
<td>StartReport</td>
<td>4</td>
<td>INFO</td>
<td>Report started</td>
</tr>
<tr>
<td>LogFileInitStatus</td>
<td>7</td>
<td>INFO</td>
<td>Log file initialization status</td>
</tr>
<tr>
<td>LogFileInitCompStatus</td>
<td>7</td>
<td>INFO</td>
<td>Log file completion status</td>
</tr>
<tr>
<td>StartReadLCP</td>
<td>10</td>
<td>INFO</td>
<td>Start read for local checkpoint</td>
</tr>
<tr>
<td>ReadLCPComplete</td>
<td>10</td>
<td>INFO</td>
<td>Read for local checkpoint completed</td>
</tr>
<tr>
<td>RunRedo</td>
<td>8</td>
<td>INFO</td>
<td>Running the redo log</td>
</tr>
<tr>
<td>RebuildIndex</td>
<td>10</td>
<td>INFO</td>
<td>Rebuilding indexes</td>
</tr>
</tbody>
</table>

### NODERESTART Events
The following events are generated when restarting a node and relate to the success or failure of the node restart process.

### Table 7.7 Events relating to restarting a node

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>NR_CopyDict</td>
<td>7</td>
<td>INFO</td>
<td>Completed copying of dictionary information</td>
</tr>
<tr>
<td>NR_CopyDistr</td>
<td>7</td>
<td>INFO</td>
<td>Completed copying distribution information</td>
</tr>
<tr>
<td>NR_CopyFragStarted</td>
<td>7</td>
<td>INFO</td>
<td>Starting to copy fragments</td>
</tr>
<tr>
<td>NR_CopyFragDone</td>
<td>10</td>
<td>INFO</td>
<td>Completed copying a fragment</td>
</tr>
<tr>
<td>NR_CopyFragCompleted</td>
<td>7</td>
<td>INFO</td>
<td>Completed copying all fragments</td>
</tr>
<tr>
<td>NodeFailCompleted</td>
<td>8</td>
<td>ALERT</td>
<td>Node failure phase completed</td>
</tr>
<tr>
<td>NODE_FAILREP</td>
<td>8</td>
<td>ALERT</td>
<td>Reports that a node has failed</td>
</tr>
<tr>
<td>ArbitState</td>
<td>6</td>
<td>INFO</td>
<td>Report whether an arbitrator is found or not; there are seven different possible outcomes when seeking an arbitrator, listed here:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Management server restarts arbitration thread [state=x]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Prepare arbitrator node x [ticket=y]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Receive arbitrator node x [ticket=y]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Started arbitrator node x [ticket=y]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Lost arbitrator node x - process failure [state=y]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Lost arbitrator node x - process exit [state=y]</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Lost arbitrator node x &lt;error msg&gt; [state=y]</td>
</tr>
<tr>
<td>ArbitResult</td>
<td>2</td>
<td>ALERT</td>
<td>Report arbitrator results; there are eight different possible results for arbitration attempts, listed here:</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Arbitration check failed: less than 1/2 nodes left</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Arbitration check succeeded: node group majority</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Arbitration check failed: missing node group</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Network partitioning: arbitration required</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Arbitration succeeded: affirmative response from node x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Arbitration failed: negative response from node x</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td>• Network partitioning: no arbitrator available</td>
</tr>
</tbody>
</table>

404
### NDB Cluster Log Events

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>GCP_TakeoverStarted</td>
<td>7</td>
<td>INFO</td>
<td>GCP takeover started</td>
</tr>
<tr>
<td>GCP_TakeoverCompleted</td>
<td>7</td>
<td>INFO</td>
<td>GCP takeover complete</td>
</tr>
<tr>
<td>LCP_TakeoverStarted</td>
<td>7</td>
<td>INFO</td>
<td>LCP takeover started</td>
</tr>
<tr>
<td>LCP_TakeoverCompleted</td>
<td>7</td>
<td>INFO</td>
<td>LCP takeover complete (state = X)</td>
</tr>
<tr>
<td>ConnectCheckStarted</td>
<td>6</td>
<td>INFO</td>
<td>Connection check started</td>
</tr>
<tr>
<td>ConnectCheckCompleted</td>
<td>6</td>
<td>INFO</td>
<td>Connection check completed</td>
</tr>
<tr>
<td>NodeFailRejected</td>
<td>6</td>
<td>ALERT</td>
<td>Node failure phase failed</td>
</tr>
</tbody>
</table>

#### 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 7.8 Events of a statistical nature**

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransReportCounters</td>
<td>8</td>
<td>INFO</td>
<td>Report transaction statistics, including numbers of transactions, commits, reads, simple reads, writes, concurrent operations, attribute information, and aborts</td>
</tr>
<tr>
<td>OperationReportCounters</td>
<td>8</td>
<td>INFO</td>
<td>Number of operations</td>
</tr>
<tr>
<td>TableCreated</td>
<td>7</td>
<td>INFO</td>
<td>Report number of tables created</td>
</tr>
<tr>
<td>JobStatistic</td>
<td>9</td>
<td>INFO</td>
<td>Mean internal job scheduling statistics</td>
</tr>
<tr>
<td>ThreadConfigLoop</td>
<td>9</td>
<td>INFO</td>
<td>Number of thread configuration loops</td>
</tr>
<tr>
<td>SendBytesStatistic</td>
<td>9</td>
<td>INFO</td>
<td>Mean number of bytes sent to node X</td>
</tr>
<tr>
<td>ReceiveBytesStatistic</td>
<td>9</td>
<td>INFO</td>
<td>Mean number of bytes received from node X</td>
</tr>
<tr>
<td>MemoryUsage</td>
<td>5</td>
<td>INFO</td>
<td>Data and index memory usage (80%, 90%, and 100%)</td>
</tr>
<tr>
<td>MTSignalStatistics</td>
<td>9</td>
<td>INFO</td>
<td>Multithreaded signals</td>
</tr>
</tbody>
</table>

#### SCHEMA Events

These events relate to NDB Cluster schema operations.

**Table 7.9 Events relating to NDB Cluster schema operations**

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CreateSchemaObject</td>
<td>8</td>
<td>INFO</td>
<td>Schema objected created</td>
</tr>
<tr>
<td>AlterSchemaObject</td>
<td>8</td>
<td>INFO</td>
<td>Schema object updated</td>
</tr>
<tr>
<td>DropSchemaObject</td>
<td>8</td>
<td>INFO</td>
<td>Schema object dropped</td>
</tr>
</tbody>
</table>

#### 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 7.10 Events relating to cluster errors and warnings

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>TransporterError</td>
<td>2</td>
<td>ERROR</td>
<td>Transporter error</td>
</tr>
<tr>
<td>TransporterWarning</td>
<td>8</td>
<td>WARNING</td>
<td>Transporter warning</td>
</tr>
<tr>
<td>MissedHeartbeat</td>
<td>8</td>
<td>WARNING</td>
<td>Node X missed heartbeat number Y</td>
</tr>
<tr>
<td>DeadDueToHeartbeat</td>
<td>8</td>
<td>ALERT</td>
<td>Node X declared “dead” due to missed heartbeat</td>
</tr>
<tr>
<td>WarningEvent</td>
<td>2</td>
<td>WARNING</td>
<td>General warning event</td>
</tr>
<tr>
<td>SubscriptionStatus</td>
<td>4</td>
<td>WARNING</td>
<td>Change in subscription status</td>
</tr>
</tbody>
</table>

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 7.11 Information events

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SentHeartbeat</td>
<td>12</td>
<td>INFO</td>
<td>Sent heartbeat</td>
</tr>
<tr>
<td>CreateLogBytes</td>
<td>11</td>
<td>INFO</td>
<td>Create log: Log part, log file, size in MB</td>
</tr>
<tr>
<td>InfoEvent</td>
<td>2</td>
<td>INFO</td>
<td>General informational event</td>
</tr>
<tr>
<td>EventBufferStatus</td>
<td>7</td>
<td>INFO</td>
<td>Event buffer status</td>
</tr>
<tr>
<td>EventBufferStatus2</td>
<td>7</td>
<td>INFO</td>
<td>Improved event buffer status information</td>
</tr>
</tbody>
</table>

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 7.12 Events relating to single user mode

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>SingleUser</td>
<td>7</td>
<td>INFO</td>
<td>Entering or exiting single user mode</td>
</tr>
</tbody>
</table>

BACKUP Events

These events provide information about backups being created or restored.

Table 7.13 Backup events

<table>
<thead>
<tr>
<th>Event</th>
<th>Priority</th>
<th>Severity Level</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>BackupStarted</td>
<td>7</td>
<td>INFO</td>
<td>Backup started</td>
</tr>
<tr>
<td>BackupStatus</td>
<td>7</td>
<td>INFO</td>
<td>Backup status</td>
</tr>
<tr>
<td>BackupCompleted</td>
<td>7</td>
<td>INFO</td>
<td>Backup completed</td>
</tr>
<tr>
<td>BackupFailedToStart</td>
<td>7</td>
<td>ALERT</td>
<td>Backup failed to start</td>
</tr>
</tbody>
</table>
### 7.3.3 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—one 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 $(2 \times \text{MaxNoOfConcurrentOperations}) + 16 + \text{MaxNoOfConcurrentTransactions}$. (For more information about these configuration parameters, see the Transaction Parameters section of Section 5.3.6, “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:

```sql
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 7.3.1, “NDB Cluster Logging Management Commands”.

### 7.4 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 7.1, “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.
Performing a Rolling Restart of an NDB Cluster

- 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.

### 7.5 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.
Performing a Rolling Restart of an NDB Cluster

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 7.8, "Online Backup of NDB Cluster", and Section 6.23, "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 4.2.4, “Installing NDB Cluster Processes as Windows Services”).

   The type of restart required is indicated in the documentation for each node configuration parameter. See Section 5.3, “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.

- All management nodes must be upgraded before any data nodes are upgraded. 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.)
NDB Cluster Single User Mode

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.

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 reread 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 7.14.38, “The ndbinfo nodes Table”.

7.6 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 7.14.38, “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.

### 7.7 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.

### 7.7.1 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 7.14 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. | • If a node other than the master fails: The creation of the node group is always rolled forward. | • If the execution of `CREATE NODEGROUP` has reached the internal commit point: When restarted, the cluster includes the new node group. Otherwise it without. |
| | • If the master fails: | • If the master fails: | • 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. |
| | • If the internal commit point has been reached: The creation of the node group is rolled forward. | • If the internal commit point has been reached: The creation of the node group is rolled forward. | |
| | • If the internal commit point has not yet been reached. The creation of the node group is rolled back. | • If the internal commit point has not yet been reached. The creation of the node group is rolled back. | |
| Table reorganization | • If a node other than the master fails: The table reorganization is always rolled forward. | • If a node other than the master fails: The table reorganization is always rolled forward. | • 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. |
| | • If the master fails: | • If the master fails: | • 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. |
| | • If the internal commit point has been reached: The table reorganization is rolled forward. | • If the internal commit point has been reached: The table reorganization is rolled forward. | |
| | • If the internal commit point has not yet been reached. The table reorganization is rolled back. | • If the internal commit point has not yet been reached. The table reorganization is rolled back. | |
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.

7.7.2 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 7.7.3, “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 \texttt{MAX\_ROWS} option, this statement does not work; instead, use \texttt{ALTER TABLE \ldots \ ALGORITHM=INPLACE MAX\_ROWS=\ldots} to reorganize such tables. You should also bear in mind that using \texttt{MAX\_ROWS} to set the number of partitions in this fashion is deprecated, and you should use \texttt{PARTITION\_BALANCE} instead; see Setting NDB\_TABLE Options, for more information.

\textbf{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 \texttt{tbl} that existed before the new nodes were added is not distributed using the new nodes until that table has been reorganized.

8. \texttt{ALTER TABLE \ldots \ 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 \texttt{OPTIMIZE TABLE} statement in the mysql client.

This works for space used by variable-width columns of in-memory NDB tables. \texttt{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 \texttt{CREATE NODEGROUP} commands in succession to add the new node groups to the cluster.

\section*{7.7.3 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.

\textbf{Starting configuration.} For purposes of illustration, we assume a minimal configuration, and that the cluster uses a \texttt{config.ini} file containing only the following information:

\begin{verbatim}
[nbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster
[nbdb]
Id = 1
HostName = 198.51.100.1
[nbdb]
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
\end{verbatim}

\textbf{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 \texttt{my.cnf} options, and that running \texttt{SHOW} in the management client produces output similar to what is shown here:
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:

```
[nbddefault]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster
[nbd]
Id = 1
HostName = 198.51.100.1
[nbd]
Id = 2
HostName = 198.51.100.2
[nbd]
Id = 3
HostName = 198.51.100.3
[nbd]
Id = 4
HostName = 198.51.100.4
[mgm]
Id = 10
api
Id=20
HostName = 198.51.100.20
[api]
Id=21
HostName = 198.51.100.21
```

---

418
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:

   ```plaintext
   ndb_mgm> 10 STOP
   Node 10 has shut down.
   Disconnecting to allow Management Server to shutdown
   shell>
   ```

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:

   ```plaintext
   shell> ndb_mgmd -f config.ini --reload
   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:

   ```plaintext
   -- 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   (8.0.24-ndb-8.0.24, Nodegroup: 0, *)
   id=2  @198.51.100.2   (8.0.24-ndb-8.0.24, 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 (8.0.24-ndb-8.0.24)
   [mysqld(API)] 2 node(s)
   id=20 @198.51.100.20 (8.0.24-ndb-8.0.24)
   id=21 @198.51.100.21 (8.0.24-ndb-8.0.24)
   ```

**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:

   ```plaintext
   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 8.0.24)
   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 8.0.24)
   ndb_mgm> Node 2: Started (version 8.0.24)
   ```

   **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.
Adding NDB Cluster Data Nodes Online: Detailed Example

**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:

```
shell> mysqladmin --user=root -p password shutdown
081208 20:19:56 mysqld_safe mysqld from pid file
/usr/local/mysql/var/tonfisk.pid ended
shell> mysqld_safe --ndbcluster --ndb-connectstring=198.51.100.10
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:

```
shell> 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  (8.0.24-ndb-8.0.24, Nodegroup: 0, *)
id=2  @198.51.100.2  (8.0.24-ndb-8.0.24, Nodegroup: 0)
id=3  @198.51.100.3  (8.0.24-ndb-8.0.24, no nodegroup)
id=4  @198.51.100.4  (8.0.24-ndb-8.0.24, no nodegroup)
[ndb_mgmd(MGM)] 1 node(s)
id=10  @198.51.100.10  (8.0.24-ndb-8.0.24)
[mysqld(API)] 2 node(s)
id=20  @198.51.100.20  (8.0.24-ndb-8.0.24)
id=21  @198.51.100.21  (8.0.24-ndb-8.0.24)
```

**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 with 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  (8.0.24-ndb-8.0.24, Nodegroup: 0, *)
id=2  @198.51.100.2  (8.0.24-ndb-8.0.24, Nodegroup: 0)
id=3  @198.51.100.3  (8.0.24-ndb-8.0.24, Nodegroup: 1)
```
Adding NDB Cluster Data Nodes Online: Detailed Example

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):

```
shell> ndb_desc -c 198.51.100.10 -d ips -p
```

--- 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
addressess Bigunsigned NULL AT=FIXED ST=MEMORY
date Bigunsigned NULL AT=FIXED ST=MEMORY

--- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex

--- Per partition info ---

<table>
<thead>
<tr>
<th>Partition</th>
<th>Row count</th>
<th>Commit count</th>
<th>Frag fixed memory</th>
<th>Frag var-sized memory</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>26086</td>
<td>26086</td>
<td>1572864</td>
<td>557056</td>
</tr>
<tr>
<td>1</td>
<td>26329</td>
<td>26329</td>
<td>1605632</td>
<td>557056</td>
</tr>
</tbody>
</table>

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 \texttt{MAX_ROWS} to set the number of partitions per table is deprecated, and you should use \texttt{PARTITION_BALANCE} instead; see \texttt{Setting NDB_Table Options}, for more information.

After issuing the statement \texttt{ALTER TABLE ips ALGORITHM=INPLACE, REORGANIZE PARTITION}, you can see using \texttt{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):

```
shell> 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
date Bigunsigned NULL AT=FIXED ST=MEMORY
-- Indexes --
PRIMARY KEY(id) - UniqueHashIndex
PRIMARY(id) - OrderedIndex
-- Per partition info --
\begin{tabular}{|c|c|c|c|c|}
\hline
Partition & Row count & Commit count & Frag fixed memory & Frag varsized memory \\
\hline
0 & 12981 & 52296 & 1572864 & 557056 \\
1 & 13236 & 52515 & 1605632 & 557056 \\
2 & 13105 & 13105 & 819200 & 294912 \\
3 & 13093 & 13093 & 819200 & 294912 \\
\hline
\end{tabular}
```

\textbf{Note}

Normally, \texttt{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 \texttt{REORGANIZE PARTITION}.

For more information, see \texttt{ALTER TABLE Statement}.

In addition, for each table, the \texttt{ALTER TABLE} statement should be followed by an \texttt{OPTIMIZE TABLE} to reclaim wasted space. You can obtain a list of all NDBCLUSTER tables using the following query against the \texttt{INFORMATION_SCHEMA.TABLES} table:

```
SELECT TABLE_SCHEMA, TABLE_NAME
FROM INFORMATION_SCHEMA.TABLES
WHERE ENGINE = 'NDBCLUSTER';
```

\textbf{Note}

The \texttt{INFORMATION_SCHEMA.TABLES.ENGINE} value for an NDB Cluster table is always \texttt{NDBCLUSTER}, regardless of whether the \texttt{CREATE TABLE} statement
Adding NDB Cluster Data Nodes Online: Detailed Example

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:

```
[nxdbd default]
DataMemory = 100M
IndexMemory = 100M
NoOfReplicas = 2
DataDir = /usr/local/mysql/var/mysql-cluster
[nxdbd]
Id = 1
HostName = 198.51.100.1
[nxdbd]
Id = 2
HostName = 198.51.100.2
[nxdbd]
Id = 3
HostName = 198.51.100.3
Nodegroup = 65536
[nxdbd]
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:

```
shell> ndbnd -cc 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:
   ```
   shell> 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.)

### 7.8 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 `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 6.23, "ndb_restore — Restore an NDB Cluster Backup"].

Starting with NDB 8.0.16, it is possible to create backups using multiple LDMs to achieve parallelism on the data nodes. See [Section 7.8.5, "Taking an NDB Backup with Parallel Data Nodes"].

#### 7.8.1 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.

### 7.8.2 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 7.8.3, “Configuration for NDB Cluster Backups”.)

The **START BACKUP** command is used to create a backup:

```
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 START BACKUP immediately, before any other options are used.

NDB 8.0.22 and later 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

Starting with NDB 8.0.24, the **PASSWORD** clause 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`
Using The NDB Cluster Management Client to Create a Backup

- `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
   ```
Using The NDB Cluster Management Client to Create a Backup

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
```

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:

```
shell> 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.
```
Configuration for NDB Cluster Backups

4. After the backup has been aborted, the management client reports this fact in a manner similar to what is shown here:

<table>
<thead>
<tr>
<th>Node 1</th>
<th>Backup 3 started from 5 has been aborted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: 1321 - Backup aborted by user request: Permanent error: User defined error</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 3</th>
<th>Backup 3 started from 5 has been aborted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: 1323 - 1323: Permanent error: Internal error</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 2</th>
<th>Backup 3 started from 5 has been aborted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: 1323 - 1323: Permanent error: Internal error</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Node 4</th>
<th>Backup 3 started from 5 has been aborted.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Error: 1323 - 1323: Permanent error: Internal error</td>
<td></td>
</tr>
</tbody>
</table>

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:

```
shell> 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.

7.8.3 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.

### 7.8.4 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.

### 7.8.5 Taking an NDB Backup with Parallel Data Nodes

Beginning with NDB 8.0.16, it is possible 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.

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 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.
In NDB 8.0.16 and later, `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 6.23.3, “Restoring from a backup taken in parallel”.

7.9 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_NDBCLUSTER=1` option to include support for NDB.

For more information about compiling NDB Cluster from source, see Section 4.1.4, “Building NDB Cluster from Source on Linux”, and Section 4.2.2, “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 5.3.9, “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:

```
shell> mysqld --ndbcluster --ndb-connectstring=ndb_mgmd.mysql.com:1186
```

See Section 5.3.3, “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:
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.

### 7.10 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 7.10.1, “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.

Beginning with NDB 8.0.19, 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 8.0.19 (and later) 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 7.14.28, “The ndbinfo diskstat Table”
- Section 7.14.29, “The ndbinfo diskstats_1sec Table”
- Section 7.14.40, “The ndbinfo pgman_time_track_stats Table”

### 7.10.1 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 NDCLUSTER;
   ```

   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 NDCLUSTER;
   ```

   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 `NDCLUSTER` 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 \texttt{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 \texttt{undo_2.log} and each file is 12 MB in size.

- \texttt{UNDO\_BUFFER\_SIZE} is limited by the amount of system memory available.
- See \texttt{CREATE LOGFILE GROUP Statement}, and \texttt{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 \texttt{ts\_1} which uses \texttt{lg\_1} as its log file group. We want the tablespace to contain two data files, named \texttt{data\_1.dat} and \texttt{data\_2.dat}, whose initial sizes are 32 MB and 48 MB, respectively. (The default value for \texttt{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 \texttt{CREATE TABLESPACE} statement creates a tablespace \texttt{ts\_1} with the data file \texttt{data\_1.dat}, and associates \texttt{ts\_1} with log file group \texttt{lg\_1}. The \texttt{ALTER TABLESPACE} adds the second data file (\texttt{data\_2.dat}).

Some items of note:

- As is the case with the \texttt{.log} file extension used in this example for undo log files, there is no special significance for the \texttt{.dat} file extension; it is used merely for easy recognition.

- When you add a data file to a tablespace using \texttt{ADD DATAFILE 'filename'}, a file with the name \texttt{filename} is created in the \texttt{ndb\_node\_id\_fs} directory within the \texttt{DataDir} of each data node in the cluster, where \texttt{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 \texttt{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 \texttt{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.

- \texttt{CREATE TABLESPACE} statements must contain an \texttt{ENGINE} clause; only tables using the same storage engine as the tablespace can be created in the tablespace. For \texttt{ALTER TABLESPACE}, an \texttt{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 \texttt{NDBCLUSTER} and \texttt{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 \texttt{CREATE TABLESPACE} and \texttt{ALTER TABLESPACE} statements, see \texttt{CREATE TABLESPACE Statement}, and \texttt{ALTER TABLESPACE Statement}. 433
Now it is possible to create a table whose unindexed columns are stored on disk using files in tablespace `ts_1`:

```sql
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 `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:

```sql
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';
```

<table>
<thead>
<tr>
<th>File</th>
<th>Type</th>
<th>Tablespace</th>
<th>Name</th>
<th>File group</th>
<th>Free</th>
<th>Total</th>
</tr>
</thead>
<tbody>
<tr>
<td>./undo_1.log</td>
<td>UNDO LOG</td>
<td>lg_1</td>
<td>NULL</td>
<td>lg_1</td>
<td>0</td>
<td>4194304</td>
</tr>
<tr>
<td>./undo_2.log</td>
<td>UNDO LOG</td>
<td>lg_1</td>
<td>NULL</td>
<td>lg_1</td>
<td>0</td>
<td>3145728</td>
</tr>
<tr>
<td>./data_1.dat</td>
<td>DATAFILE</td>
<td>ts_1</td>
<td>NULL</td>
<td>lg_1</td>
<td>32</td>
<td>32</td>
</tr>
<tr>
<td>./data_2.dat</td>
<td>DATAFILE</td>
<td>ts_1</td>
<td>NULL</td>
<td>lg_1</td>
<td>48</td>
<td>48</td>
</tr>
</tbody>
</table>
```

For more information and examples, see 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:

```sql
mysql> CREATE TABLE ti (
            -> c1 INT PRIMARY KEY,
            -> c2 INT,
            -> c3 INT,
            -> c4 INT
```
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`:

```sql
CREATE INDEX i1 ON ti(c2);
```

Query OK, 0 rows affected (2.72 sec)
Records: 0  Duplicates: 0  Warnings: 0
```

```sql
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.

```sql
SHOW CREATE TABLE ti\G
```

Table: ti

```
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:

```bash
./ndb_desc --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
```

435
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:

```sql
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.

7.10.2 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 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 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.

7.11 Online Operations with ALTER TABLE in NDB Cluster

MySQL NDB Cluster 8.0 supports online table schema changes using the standard `ALTER TABLE` syntax employed by the MySQL Server (`ALGORITHM=DEFAULT|INPLACE|COPY`), and described elsewhere.

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.

Previously, columns of NDB tables could not be renamed online; this limitation is removed in NDB 8.0.18.

Currently you cannot add disk-based columns 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:

```sql
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:

```sql
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:

```sql
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:

```sql
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;
*************************** 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;
*************************** 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
```

**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 7.7, “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:

```sql
mysql> CREATE TABLE t2 (  
    > c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
    > ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)
```

```sql
mysql> ALTER TABLE t2
    > ADD COLUMN c2 INT,
    > ADD COLUMN c3 INT,
    > ALGORITHM=INPLACE;
Query OK, 0 rows affected, 2 warnings (0.93 sec)
```

```sql
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)
```

- 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):

```sql
mysql> CREATE TABLE t2 (  
    > c1 INT NOT NULL AUTO_INCREMENT PRIMARY KEY
    > ) ENGINE=NDB;
Query OK, 0 rows affected (1.44 sec)
```

```sql
mysql> ALTER TABLE t2
    > ADD COLUMN c2 INT,
    > ADD COLUMN c3 INT,
    > ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 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:

```sql
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)
```

```sql
mysql> ALTER TABLE t3 ADD COLUMN c2 INT, ALGORITHM=INPLACE;
Query OK, 0 rows affected, 1 warning (0.78 sec)
```
Distributed MySQL Privileges with NDB_STORED_USER

NDB 8.0.18 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. To verify that privileges are shared, use the ndb_select_all utility supplied with the NDB Cluster distribution, as shown here (some output wrapped to preserve formatting):

```
shell> ndb_select_all -d mysql ndb_sql_metadata
```

```
type name seq note sql_ddl_text
11  "'jon'@'localhost'" 0 4  "CREATE USER 'jon'@'localhost'
   IDENTIFIED WITH 'caching_sha2_password' AS
   'SA00S$[B];3f!t\"E.Fy2A5k5DqHsWi8vUYTiMw0oYeb1$P2otFRPjVvZyT5a5b0' REQUIRE
   none PASSWORD EXPIRE DEFAULT ACCOUNT UNLOCK PASSWORD HISTORY DEFAULT PASSWORD
   REUSE INTERVAL DEFAULT PASSWORD REQUIRE CURRENT DEFAULT"
12  "'jon'@'localhost'" 3 [NULL] "GRANT ALL PRIVILEGES ON 'test'.* TO 'jon'@'localhost'"
12  "'jon'@'localhost'" 2 [NULL] "GRANT ALL PRIVILEGES ON 'mydb'.* TO 'jon'@'localhost'"
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.

You should be aware that changes to users with NDB_STORED_USER are distributed asynchronously. Because distributed schema change operations are performed synchronously, the next distributed schema change following a change to any distributed 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
NDB API Statistics Counters and Variables

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 converting the MySQL privilege tables to use the NDB storage engine, as implemented 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 4.7, “Upgrading and Downgrading NDB Cluster”.

7.13 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_session   | 0        |
| Ndb_api_wait_scan_result_count_session     | 0        |
| Ndb_api_wait_meta_request_count_session    | 0        |
| Ndb_api_wait_nanos_count_session           | 0        |
| Ndb_api_bytes_sent_count_session           | 0        |
| Ndb_api_bytes_received_count_session       | 0        |
| Ndb_api_trans_start_count_session          | 0        |
| Ndb_api_trans_commit_count_session         | 0        |
| Ndb_api_trans_abort_count_session          | 0        |
| Ndb_api_trans_close_count_session          | 0        |
| Ndb_api_pk_op_count_session                | 0        |
| Ndb_api_uk_op_count_session                | 0        |
| Ndb_api_table_scan_count_session           | 0        |
| Ndb_api_range_scan_count_session           | 0        |
| Ndb_api_pruned_scan_count_session          | 0        |
| Ndb_api_scan_batch_count_session           | 0        |
| Ndb_api_read_row_count_session             | 0        |
| Ndb_api_trans_local_read_row_count_session | 0        |
| Ndb_api_event_data_countInjector           | 0        |
| Ndb_api_event_nodata_countInjector         | 0        |
| Ndb_api_event_bytes_countInjector          | 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_wait_exec_complete_count           | 2        |
+--------------------------------------------+----------+
```
## NDB API Statistics Counters and Variables

<table>
<thead>
<tr>
<th>NDB_API_WAIT_SCAN_RESULT_COUNT</th>
<th>3</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDB_API_WAIT_META_REQUEST_COUNT</td>
<td>27</td>
</tr>
<tr>
<td>NDB_API_WAIT_NANOS_COUNT</td>
<td>45612023</td>
</tr>
<tr>
<td>NDB_API_BYTES_SENT_COUNT</td>
<td>992</td>
</tr>
<tr>
<td>NDB_API_BYTES_RECEIVED_COUNT</td>
<td>9640</td>
</tr>
<tr>
<td>NDB_API_TRANS_START_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_TRANS_COMMIT_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_TRANS_ABORT_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_CLOSE_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_PK_OP_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_UK_OP_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TABLE_SCAN_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_RANGE_SCAN_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_PRUNED_SCAN_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_SCAN_BATCH_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_READ_ROW_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_TRANS_LOCAL_READ_ROW_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_EVENT_DATA_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_NONDATA_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_BYTES_COUNT</td>
<td>0</td>
</tr>
</tbody>
</table>

60 rows in set (0.02 sec)

These status variables are also available from the Performance Schema `session_status` and `global_status` tables, as shown here:

```sql
mysql> SELECT * FROM performance_schema.session_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
+--------------------------------------------+----------------+----------------+
| VARIABLE_NAME                              | VARIABLE_VALUE |
| NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION   | 2              |
| NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION     | 0              |
| NDB_API_WAIT_META_REQUEST_COUNT_SESSION    | 1              |
| NDB_API_WAIT_NANOS_COUNT_SESSION           | 83144375       |
| NDB_API_BYTES_SENT_COUNT_SESSION           | 68             |
| NDB_API_BYTES_RECEIVED_COUNT_SESSION       | 84             |
| NDB_API_TRANS_START_COUNT_SESSION          | 1              |
| NDB_API_TRANS_COMMIT_COUNT_SESSION         | 1              |
| NDB_API_TRANS_ABORT_COUNT_SESSION          | 0              |
| NDB_API_TRANS_CLOSE_COUNT_SESSION          | 1              |
| NDB_API_PK_OP_COUNT_SESSION                | 1              |
| NDB_API_UK_OP_COUNT_SESSION                | 0              |
| NDB_API_TABLE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_RANGE_SCAN_COUNT_SESSION           | 0              |
| NDB_API_PRUNED_SCAN_COUNT_SESSION          | 0              |
| NDB_API_SCAN_BATCH_COUNT_SESSION           | 0              |
| NDB_API_READ_ROW_COUNT_SESSION             | 1              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION | 1              |
| NDB_API_EVENT_DATA_COUNT_INJECTOR          | 0              |
| NDB_API_EVENT_NONDATA_COUNT_INJECTOR       | 0              |
| NDB_API_EVENT_BYTES_COUNT_INJECTOR         | 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_WAIT_EXEC_COMPLETE_COUNT           | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT             | 3              |
```

442
### NDB API Statistics Counters and Variables

<table>
<thead>
<tr>
<th>Variable_Name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDB_API_WAIT_META_REQUEST_COUNT</td>
<td>28</td>
</tr>
<tr>
<td>NDB_API_WAIT_NANOS_COUNT</td>
<td>53756398</td>
</tr>
<tr>
<td>NDB_API_BYTES_SENT_COUNT</td>
<td>1060</td>
</tr>
<tr>
<td>NDB_API_BYTES_RECEIVED_COUNT</td>
<td>9724</td>
</tr>
<tr>
<td>NDB_API_TRANS_START_COUNT</td>
<td>3</td>
</tr>
<tr>
<td>NDB_API_TRANS_COMMIT_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_TRANS_ABORT_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_CLOSE_COUNT</td>
<td>3</td>
</tr>
<tr>
<td>NDB_API_PK_OP_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_UK_OP_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TABLE_SCAN_COUNT</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_RANGE_SCAN_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_PRUNED_SCAN_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_SCAN_BATCH_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_READ_ROW_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_TRANS_LOCAL_READ_ROW_COUNT</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_EVENT_DATA_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_NONDATA_COUNT</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_BYTES_COUNT</td>
<td>0</td>
</tr>
</tbody>
</table>

60 rows in set (0.00 sec)

```sql
mysql> SELECT * FROM performance_schema.global_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%';
```

<table>
<thead>
<tr>
<th>VARIABLE_NAME</th>
<th>VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>NDB_API_WAIT_EXEC_COMPLETE_COUNT_SESSION</td>
<td>2</td>
</tr>
<tr>
<td>NDB_API_WAIT_SCAN_RESULT_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_META_REQUEST_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_WAIT_NANOS_COUNT_SESSION</td>
<td>8144375</td>
</tr>
<tr>
<td>NDB_API_BYTES_SENT_COUNT_SESSION</td>
<td>68</td>
</tr>
<tr>
<td>NDB_API_BYTES_RECEIVED_COUNT_SESSION</td>
<td>84</td>
</tr>
<tr>
<td>NDB_API_TRANS_START_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_TRANS_COMMIT_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_TRANS_ABORT_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_CLOSE_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_PK_OP_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_UK_OP_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TABLE_SCAN_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_RANGE_SCAN_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_PRUNED_SCAN_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_SCAN_BATCH_COUNT_SESSION</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_READ_ROW_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SESSION</td>
<td>1</td>
</tr>
<tr>
<td>NDB_API_EVENT_DATA_COUNT_INJECTOR</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_NONDATA_COUNT_INJECTOR</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_EVENT_BYTES_COUNT_INJECTOR</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_EXEC_COMPLETE_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_SCAN_RESULT_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_META_REQUEST_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_NANOS_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_BYTES_SENT_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_BYTES_RECEIVED_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_START_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_COMMIT_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_ABORT_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_CLOSE_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_PK_OP_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_UK_OP_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TABLE_SCAN_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_RANGE_SCAN_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_PRUNED_SCAN_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_SCAN_BATCH_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_READ_ROW_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_TRANS_LOCAL_READ_ROW_COUNT_SLAVE</td>
<td>0</td>
</tr>
<tr>
<td>NDB_API_WAIT_EXEC_COMPLETE_COUNT</td>
<td>4</td>
</tr>
<tr>
<td>NDB_API_WAIT_SCAN_RESULT_COUNT</td>
<td>3</td>
</tr>
<tr>
<td>NDB_API_WAIT_META_REQUEST_COUNT</td>
<td>28</td>
</tr>
<tr>
<td>NDB_API_WAIT_NANOS_COUNT</td>
<td>53756398</td>
</tr>
<tr>
<td>NDB_API_BYTES_SENT_COUNT</td>
<td>1060</td>
</tr>
<tr>
<td>NDB_API_BYTES_RECEIVED_COUNT</td>
<td>9724</td>
</tr>
</tbody>
</table>
Each Ndb object has its own counters. NDB API applications can read the values of the counters for use in optimization or monitoring. For multithreaded clients which use more than one Ndb object concurrently, it is also possible to obtain a summed view of counters from all Ndb objects belonging to a given Ndb_cluster_connection.

Four sets of these counters are exposed. One set applies to the current session only; the other 3 are global. This is in spite of the fact that their values can be obtained as either session or global status variables in the mysql client. This means that specifying the SESSION or GLOBAL keyword with SHOW STATUS has no effect on the values reported for NDB API statistics status variables, and the value for each of these variables is the same whether the value is obtained from the equivalent column of the session_status or the global_status table.

- **Session counters (session specific)**

  Session counters relate to the Ndb objects in use by (only) the current session. Use of such objects by other MySQL clients does not influence these counts.

  In order to minimize confusion with standard MySQL session variables, we refer to the variables that correspond to these NDB API session counters as "_session variables", with a leading underscore.

- **Replica counters (global)**

  This set of counters relates to the Ndb objects used by the replica SQL thread, if any. If this mysqld does not act as a replica, or does not use NDB tables, then all of these counts are 0.

  We refer to the related status variables as "_slave variables" (with a leading underscore).

- **Injector counters (global)**

  Injector counters relate to the Ndb object used to listen to cluster events by the binary log injector thread. Even when not writing a binary log, mysqld processes attached to an NDB Cluster continue to listen for some events, such as schema changes.

  We refer to the status variables that correspond to NDB API injector counters as "_injector variables" (with a leading underscore).

- **Server (Global) counters (global)**

  This set of counters relates to all Ndb objects currently used by this mysqld. This includes all MySQL client applications, the replica SQL thread (if any), the binary log injector, and the NDB utility thread.

  We refer to the status variables that correspond to these counters as "global variables" or "mysqld-level variables".
You can obtain values for a particular set of variables by additionally filtering for the substring _session, _slave, or _injector in the variable name (along with the common prefix Ndb_api). For _session variables, this can be done as shown here:

```sql
mysql> SHOW STATUS LIKE 'ndb_api%session';
+--------------------------------------------+---------+
| Variable_name                              | Value   |
+--------------------------------------------+---------+
| Ndb_api_wait_exec_complete_count_session   | 2       |
| Ndb_api_wait_scan_result_count_session     | 0       |
| Ndb_api_wait_meta_request_count_session    | 1       |
| Ndb_api_wait_nanos_count_session           | 8144375 |
| Ndb_api_bytes_sent_count_session           | 68      |
| Ndb_api_bytes_received_count_session       | 84      |
| Ndb_api_trans_start_count_session          | 1       |
| Ndb_api_trans_commit_count_session         | 1       |
| Ndb_api_trans_abort_count_session          | 0       |
| Ndb_api_trans_close_count_session          | 1       |
| Ndb_api_pk_op_count_session                | 1       |
| Ndb_api_uk_op_count_session                | 0       |
| Ndb_api_table_scan_count_session           | 0       |
| Ndb_api_range_scan_count_session           | 0       |
| Ndb_api_pruned_scan_count_session          | 0       |
| Ndb_api_scan_batch_count_session           | 0       |
| Ndb_api_read_row_count_session             | 1       |
| Ndb_api_trans_local_read_row_count_session | 1       |
+--------------------------------------------+---------+
18 rows in set (0.50 sec)
```

To obtain a listing of the NDB API mysqld-level status variables, filter for variable names beginning with ndb_api and ending in _count, like this:

```sql
mysql> SELECT * FROM performance_schema.session_status
-> WHERE VARIABLE_NAME LIKE 'ndb_api%count';
+------------------------------------+----------------+
| VARIABLE_NAME                      | VARIABLE_VALUE |
+------------------------------------+----------------+
| NDB_API_WAIT_EXEC_COMPLETE_COUNT   | 4              |
| NDB_API_WAIT_SCAN_RESULT_COUNT     | 3              |
| NDB_API_WAIT_META_REQUEST_COUNT    | 28             |
| NDB_API_WAIT_NANOS_COUNT           | 53756398       |
| NDB_API_BYTES_SENT_COUNT           | 1060           |
| NDB_API_BYTES_RECEIVED_COUNT       | 9724           |
| NDB_API_TRANS_START_COUNT          | 3              |
| NDB_API_TRANS_COMMIT_COUNT         | 2              |
| NDB_API_TRANS_ABORT_COUNT          | 0              |
| NDB_API_TRANS_CLOSE_COUNT          | 3              |
| NDB_API_PK_OP_COUNT                | 2              |
| NDB_API_UK_OP_COUNT                | 0              |
| NDB_API_TABLE_SCAN_COUNT           | 1              |
| NDB_API_RANGE_SCAN_COUNT           | 0              |
| NDB_API_PRUNED_SCAN_COUNT          | 0              |
| NDB_API_SCAN_BATCH_COUNT           | 0              |
| NDB_API_READ_ROW_COUNT             | 2              |
| NDB_API_TRANS_LOCAL_READ_ROW_COUNT | 2              |
| NDB_API_EVENT_DATA_COUNT           | 0              |
| NDB_API_EVENT_NONDATA_COUNT        | 0              |
| NDB_API_EVENT_BYTES_COUNT          | 0              |
+------------------------------------+----------------+
21 rows in set (0.09 sec)
```

Not all counters are reflected in all 4 sets of status variables. For the event counters DataEventsRecvdCount, NondataEventsRecvdCount, and EventBytesRecvdCount, only _injector and mysqld-level NDB API status variables are available:

```sql
mysql> SHOW STATUS LIKE 'ndb_api%event%';
+--------------------------------------+-------+
| Variable_name                        | Value |
+--------------------------------------+-------+
| Ndb_api_event_data_count_injector    | 0     |
| Ndb_api_event_nondata_count_injector | 0     |
+--------------------------------------+-------+
```

445
NDB API Statistics Counters and Variables

```sql
| Ndb_api_event_bytes_count_injector | 0     |
| Ndb_api_event_data_count           | 0     |
| Ndb_api_event_nondata_count        | 0     |
| Ndb_api_event_bytes_count          | 0     |
+------------------------------------+-------+
6 rows in set (0.00 sec)
```

_injector status variables are not implemented for any other NDB API counters, as shown here:

```sql
mysql> SHOW STATUS LIKE 'ndb_api%injector%';
+--------------------------------------+-------+
| Variable_name                        | Value |
|--------------------------------------+-------|
| Ndb_api_event_data_count_injector    | 0     |
| Ndb_api_event_nondata_count_injector | 0     |
| Ndb_api_event_bytes_count_injector   | 0     |
+--------------------------------------+-------+
3 rows in set (0.00 sec)
```

The names of the status variables can easily be associated with the names of the corresponding counters. Each NDB API statistics counter is listed in the following table with a description as well as the names of any MySQL server status variables corresponding to this counter.

**Table 7.15 NDB API statistics counters**

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>Status Variables (by statistic type):</th>
</tr>
</thead>
</table>
| WaitExecCompleteCount | Number of times thread has been blocked while waiting for execution of an operation to complete. Includes all `execute()` calls as well as implicit executes for blob operations and auto-increment not visible to clients. | • Ndb_api_wait_exec_complete_count_session
                         |                                                                                   | • Ndb_api_wait_exec_complete_count_slave
                         |                                                                                   | [none]                                                                                   | • Ndb_api_wait_exec_complete_count |
| WaitScanResultCount   | Number of times thread has been blocked while waiting for a scan-based signal, such waiting for additional results, or for a scan to close. | • Ndb_api_wait_scan_result_count_session
                         |                                                                                   | • Ndb_api_wait_scan_result_count_slave
                         |                                                                                   | [none]                                                                                   | • Ndb_api_wait_scan_result_count |
| WaitMetaRequestCount  | Number of times thread has been blocked waiting for a metadata-based signal; this can occur when waiting for a DDL operation or for an epoch to be started (or ended). | • Ndb_api_wait_meta_request_count_session
                         |                                                                                   | • Ndb_api_wait_meta_request_count_slave
                         |                                                                                   | [none]                                                                                   | • Ndb_api_wait_meta_request_count |
| WaitNanosCount        | Total time (in nanoseconds) spent waiting for some type of signal from the data nodes. | • Ndb_api_wait_nanos_count_session
                         |                                                                                   | • Ndb_api_wait_nanos_count_slave
                         |                                                                                   | [none]                                                                                   | • Ndb_api_wait_nanos_count      |

446
<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>Status Variables (by statistic type):</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>• Session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Slave (replica)</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Injector</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Server</td>
</tr>
<tr>
<td>BytesSentCount</td>
<td>Amount of data (in bytes) sent to the data nodes</td>
<td>• Ndb_api_bytes_sent_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_bytes_sent_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_bytes_sent_count</td>
</tr>
<tr>
<td>BytesRecvdCount</td>
<td>Amount of data (in bytes) received from the data nodes</td>
<td>• Ndb_api_bytes_received_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_bytes_received_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_bytes_received_count</td>
</tr>
<tr>
<td>TransStartCount</td>
<td>Number of transactions started.</td>
<td>• Ndb_api_trans_start_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_start_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_start_count</td>
</tr>
<tr>
<td>TransCommitCount</td>
<td>Number of transactions committed.</td>
<td>• Ndb_api_trans_commit_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_commit_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_commit_count</td>
</tr>
<tr>
<td>TransAbortCount</td>
<td>Number of transactions aborted.</td>
<td>• Ndb_api_trans_abort_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_abort_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_abort_count</td>
</tr>
<tr>
<td>TransCloseCount</td>
<td>Number of transactions aborted. (This value may be greater than the sum of</td>
<td>• Ndb_api_trans_close_count_session</td>
</tr>
<tr>
<td></td>
<td>TransCommitCount and TransAbortCount.)</td>
<td>• Ndb_api_trans_close_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>• Ndb_api_trans_close_count</td>
</tr>
<tr>
<td>PkOpCount</td>
<td>Number of operations based on or using primary keys. This count includes</td>
<td>• Ndb_api_pk_op_count_session</td>
</tr>
<tr>
<td></td>
<td>blob-part table operations, implicit unlocking operations, and auto-increment</td>
<td>• Ndb_api_pk_op_count_slave</td>
</tr>
<tr>
<td></td>
<td>operations, as well as primary key operations normally visible to MySQL</td>
<td>• [none]</td>
</tr>
<tr>
<td></td>
<td>clients.</td>
<td>• Ndb_api_pk_op_count</td>
</tr>
<tr>
<td>Counter Name</td>
<td>Description</td>
<td>Status Variables (by statistic type):</td>
</tr>
<tr>
<td>-------------------</td>
<td>-----------------------------------------------------------------------------</td>
<td>----------------------------------------</td>
</tr>
<tr>
<td>UkOpCount</td>
<td>Number of operations based on or using unique keys.</td>
<td>Ndb_api_uk_op_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_uk_op_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_uk_op_count</td>
</tr>
<tr>
<td>TableScanCount</td>
<td>Number of table scans that have been started. This includes scans of internal tables.</td>
<td>Ndb_api_table_scan_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_table_scan_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_table_scan_count</td>
</tr>
<tr>
<td>RangeScanCount</td>
<td>Number of range scans that have been started.</td>
<td>Ndb_api_range_scan_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_range_scan_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_range_scan_count</td>
</tr>
<tr>
<td>PrunedScanCount</td>
<td>Number of scans that have been pruned to a single partition.</td>
<td>Ndb_api_pruned_scan_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_pruned_scan_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_pruned_scan_count</td>
</tr>
<tr>
<td>ScanBatchCount</td>
<td>Number of batches of rows received. (A batch in this context is a set of scan results from a single fragment.)</td>
<td>Ndb_api_scan_batch_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_scan_batch_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_scan_batch_count</td>
</tr>
<tr>
<td>ReadRowCount</td>
<td>Total number of rows that have been read. Includes rows read using primary key, unique key, and scan operations.</td>
<td>Ndb_api_read_row_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_read_row_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_read_row_count</td>
</tr>
<tr>
<td>TransLocalReadRowCount</td>
<td>Number of rows read from the data same node on which the transaction was being run.</td>
<td>Ndb_api_trans_local_read_row_count_session</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_trans_local_read_row_count_slave</td>
</tr>
<tr>
<td></td>
<td></td>
<td>[none]</td>
</tr>
<tr>
<td></td>
<td></td>
<td>Ndb_api_trans_local_read_row_count</td>
</tr>
</tbody>
</table>
## NDB API Statistics Counters and Variables

<table>
<thead>
<tr>
<th>Counter Name</th>
<th>Description</th>
<th>Status Variables (by statistic type):</th>
</tr>
</thead>
<tbody>
<tr>
<td>DataEventsRecvdCount</td>
<td>Number of row change events received.</td>
<td>[none]</td>
</tr>
<tr>
<td>NondataEventsRecvdCount</td>
<td>Number of events received, other than row change events.</td>
<td>[none]</td>
</tr>
<tr>
<td>EventBytesRecvdCount</td>
<td>Number of bytes of events received.</td>
<td>[none]</td>
</tr>
</tbody>
</table>

To see all counts of committed transactions—that is, all `TransCommitCount` counter status variables—you can filter the results of `SHOW STATUS` for the substring `trans_commit_count`, like this:

```
mysql> SHOW STATUS LIKE '%trans_commit_count%';
+------------------------------------+-------+
| Variable_name                      | Value |
+------------------------------------+-------+
| Ndb_api_trans_commit_count_session | 1     |
| Ndb_api_trans_commit_count_slave   | 0     |
| Ndb_api_trans_commit_count         | 2     |
+------------------------------------+-------+
3 rows in set (0.00 sec)
```

From this you can determine that 1 transaction has been committed in the current `mysql` client session, and 2 transactions have been committed on this `mysqld` since it was last restarted.

You can see how various NDB API counters are incremented by a given SQL statement by comparing the values of the corresponding `_session` status variables immediately before and after performing the statement. In this example, after getting the initial values from `SHOW STATUS`, we create in the `test` database an NDB table, named `t`, that has a single column:

```
mysql> SHOW STATUS LIKE 'ndb_api%session%';
+--------------------------------------------+--------+
| Variable_name                              | Value  |
+--------------------------------------------+--------+
| Ndb_api_wait_exec_complete_count_session   | 2      |
| Ndb_api_wait_scan_result_count_session     | 0      |
| Ndb_api_wait_meta_request_count_session    | 3      |
| Ndb_api_wait_nanos_count_session           | 820705 |
| Ndb_api_bytes_sent_count_session           | 132    |
| Ndb_api_bytes_received_count_session       | 372    |
| Ndb_api_trans_start_count_session          | 1      |
| Ndb_api_trans_commit_count_session         | 1      |
+--------------------------------------------+--------+
```

449
### NDB API Statistics Counters and Variables

<table>
<thead>
<tr>
<th>Variable_name</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ndb_api_trans_abort_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_trans_close_count_session</td>
<td>1</td>
</tr>
<tr>
<td>Ndb_api_pk_op_count_session</td>
<td>1</td>
</tr>
<tr>
<td>Ndb_api_uk_op_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_table_scan_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_range_scan_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_pruned_scan_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_scan_batch_count_session</td>
<td>0</td>
</tr>
<tr>
<td>Ndb_api_read_row_count_session</td>
<td>1</td>
</tr>
<tr>
<td>Ndb_api_trans_local_read_row_count_session</td>
<td>1</td>
</tr>
</tbody>
</table>

18 rows in set (0.00 sec)

mysql> USE test;
Database changed

mysql> CREATE TABLE t (c INT) ENGINE NDBCLUSTER;
Query OK, 0 rows affected (0.85 sec)

Now you can execute a new `SHOW STATUS` statement and observe the changes, as shown here (with the changed rows highlighted in the output):

```
mysql> SHOW STATUS LIKE 'ndb_api\%session\%';
+--------------------------------------------+-----------+
| Variable_name                              | Value     |
+--------------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session   | 8         |
| Ndb_api_wait_scan_result_count_session     | 0         |
| Ndb_api_wait_meta_request_count_session    | 17        |
| Ndb_api_wait_nanos_count_session           | 706871709 |
| Ndb_api_bytes_sent_count_session           | 2376      |
| Ndb_api_bytes_received_count_session       | 3844      |
| Ndb_api_trans_start_count_session          | 4         |
| Ndb_api_trans_commit_count_session         | 4         |
| Ndb_api_transAbort_count_session           | 0         |
| Ndb_api_transClose_count_session           | 4         |
| Ndb_api_pk_op_count_session                | 6         |
| Ndb_api_uk_op_count_session                | 0         |
| Ndb_api_table_scan_count_session           | 0         |
| Ndb_api_range_scan_count_session           | 0         |
| Ndb_api_pruned_scan_count_session          | 0         |
| Ndb_api_scan_batch_count_session           | 0         |
| Ndb_api_read_row_count_session             | 2         |
| Ndb_api_trans_local_read_row_count_session | 1         |
+--------------------------------------------+-----------+
18 rows in set (0.00 sec)
```

Similarly, you can see the changes in the NDB API statistics counters caused by inserting a row into `t`:

Insert the row, then run the same `SHOW STATUS` statement used in the previous example, as shown here:

```
mysql> INSERT INTO t VALUES (100);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> SHOW STATUS LIKE 'ndb_api\%session\%';
+--------------------------------------------+-----------+
| Variable_name                              | Value     |
+--------------------------------------------+-----------+
| Ndb_api_wait_exec_complete_count_session   | 11        |
| Ndb_api_wait_scan_result_count_session     | 6         |
| Ndb_api_wait_meta_request_count_session    | 20        |
| Ndb_api_wait_nanos_count_session           | 707370418 |
| Ndb_api_bytes_sent_count_session           | 2724      |
| Ndb_api_bytes_received_count_session       | 4116      |
| Ndb_api_trans_start_count_session          | 7         |
| Ndb_api_trans_commit_count_session         | 6         |
| Ndb_api_transAbort_count_session           | 0         |
| Ndb_api_transClose_count_session           | 7         |
| Ndb_api_pk_op_count_session                | 8         |
| Ndb_api_uk_op_count_session                | 0         |
| Ndb_api_table_scan_count_session           | 1         |
| Ndb_api_range_scan_count_session           | 0         |
| Ndb_api_pruned_scan_count_session          | 0         |
| Ndb_api_scan_batch_count_session           | 0         |
+--------------------------------------------+-----------+
18 rows in set (0.00 sec)
```
We can make a number of observations from these results:

- Although we created `t` with no explicit primary key, 5 primary key operations were performed in doing so (the difference in the “before” and “after” values of `Ndb_api_pk_op_count_session`, or 6 minus 1). This reflects the creation of the hidden primary key that is a feature of all tables using the NDB storage engine.

- By comparing successive values for `Ndb_api_wait_nanos_count_session`, we can see that the NDB API operations implementing the `CREATE TABLE` statement waited much longer (706871709 - 820705 = 706051004 nanoseconds, or approximately 0.7 second) for responses from the data nodes than those executed by the `INSERT` (707370418 - 706871709 = 498709 ns or roughly .0005 second). The execution times reported for these statements in the `mysql` client correlate roughly with these figures.

On platforms without sufficient (nanosecond) time resolution, small changes in the value of the `WaitNanosCount` NDB API counter due to SQL statements that execute very quickly may not always be visible in the values of `Ndb_api_wait_nanos_count_session`, `Ndb_api_wait_nanos_count_slave`, or `Ndb_api_wait_nanos_count`.

- The `INSERT` statement incremented both the `ReadRowCount` and `TransLocalReadRowCount` NDB API statistics counters, as reflected by the increased values of `Ndb_api_read_row_count_session` and `Ndb_api_trans_local_read_row_count_session`.

7.14 ndbinfo: The NDB Cluster Information Database

ndbinfo is a database containing information specific to NDB Cluster. This database contains a number of tables, each providing a different sort of data about NDB Cluster node status, resource usage, and operations. You can find more detailed information about each of these tables in the next several sections.

ndbinfo is included with NDB Cluster support in the MySQL Server; no special compilation or configuration steps are required; the tables are created by the MySQL Server when it connects to the cluster. You can verify that ndbinfo support is active in a given MySQL Server instance using `SHOW PLUGINS`; if ndbinfo support is enabled, you should see a row containing ndbinfo in the Name column and ACTIVE in the Status column, as shown here (emphasized text):

```
mysql> SHOW PLUGINS;
+----------------------------------+--------+--------------------+---------+---------+
| Name                             | Status | Type               | Library | License |
+----------------------------------+--------+--------------------+---------+---------+
| binlog                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password            | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| sha256_password                  | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| MRG_MYISAM                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| CSV                              | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| InnoDB                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                     | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS                | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET                 | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX             | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_PER_INDEX_RESET       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_BUFFER_PAGE               | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
```
You can also do this by checking the output of `SHOW ENGINES` for a line including `ndbinfo` in the `Engine` column and `YES` in the `Support` column, as shown here (emphasized text):

```sql
mysql> SHOW ENGINES\G
*************************** 1. row ***************************
Engine: ndbcluster
Support: YES
Comment: Clustered, fault-tolerant tables
Transactions: YES
XA: NO
Savepoints: NO
*************************** 2. row ***************************
Engine: CSV
Support: YES
Comment: CSV storage engine
Transactions: NO
XA: NO
Savepoints: NO
*************************** 3. row ***************************
Engine: InnoDB
Support: DEFAULT
Comment: Supports transactions, row-level locking, and foreign keys
Transactions: YES
XA: YES
Savepoints: YES
*************************** 4. row ***************************
Engine: BLACKHOLE
Support: YES
Comment: /dev/null storage engine (anything you write to it disappears)
Transactions: NO
XA: NO
Savepoints: NO
*************************** 5. row ***************************
Engine: MyISAM
Support: YES
Comment: MyISAM storage engine
Transactions: NO
XA: NO
Savepoints: NO
*************************** 6. row ***************************
Engine: MRG_MYISAM
```
If `ndbinfo` support is enabled, then you can access `ndbinfo` using SQL statements in `mysql` or another MySQL client. For example, you can see `ndbinfo` listed in the output of `SHOW DATABASES`, as shown here (emphasized text):

```
mysql> SHOW DATABASES;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| ndbinfo            |
| performance_schema |
| sys                |
+--------------------+
5 rows in set (0.04 sec)
```

If the `mysqld` process was not started with the `--ndbcluster` option, `ndbinfo` is not available and is not displayed by `SHOW DATABASES`. If `mysqld` was formerly connected to an NDB Cluster but the cluster becomes unavailable (due to events such as cluster shutdown, loss of network connectivity, and so forth), `ndbinfo` and its tables remain visible, but an attempt to access any tables (other than `blocks` or `config_params`) fails with `Got error 157 'Connection to NDB failed' from NDBINFO`.

With the exception of the `blocks` and `config_params` tables, what we refer to as `ndbinfo` “tables” are actually views generated from internal NDB tables not normally visible to the MySQL Server.

All `ndbinfo` tables are read-only, and are generated on demand when queried. Because many of them are generated in parallel by the data nodes while other are specific to a given SQL node, they are not guaranteed to provide a consistent snapshot.

In addition, pushing down of joins is not supported on `ndbinfo` tables; so joining large `ndbinfo` tables can require transfer of a large amount of data to the requesting API node, even when the query makes use of a `WHERE` clause.
**ndbinfo** tables are not included in the query cache. (Bug #59831)

You can select the **ndbinfo** database with a `USE` statement, and then issue a `SHOW TABLES` statement to obtain a list of tables, just as for any other database, like this:

```sql
mysql> USE ndbinfo;
Database changed
mysql> SHOW TABLES;
```

```
+---------------------------------+
| Tables_in_ndbinfo               |
+---------------------------------+
| arbitrator_validity_detail      |
| arbitrator_validity_summary     |
| blocks                          |
| cluster_locks                   |
| cluster_operations              |
| cluster_transactions           |
| config_nodes                    |
| config_params                   |
| config_values                   |
| counters                        |
| cpustat                         |
| cpustat_lsec                    |
| cpustat_20sec                   |
| cpustat_50ms                    |
| dict_obj_info                   |
| dict_obj_types                  |
| disk_write_speed_aggregate      |
| disk_write_speed_aggregate_node |
| disk_write_speed_base           |
| diskpagebuffer                  |
| error_messages                  |
| locks_per_fragment              |
| logbuffers                      |
| logspaces                       |
| membership                      |
| memory_per_fragment             |
| memoryusage                     |
| nodes                           |
| operations_per_fragment         |
| processes                       |
| resources                       |
| restart_info                    |
| server_locks                    |
| server_operations               |
| server_transactions            |
| table_distribution_status       |
| table_fragment                  |
| table_info                      |
| table_replicas                  |
| tc_time_track_stats             |
| threadblocks                    |
| threads                         |
| threadstat                      |
| transporters                    |
+---------------------------------+
44 rows in set (0.00 sec)
```

In NDB 8.0, all **ndbinfo** tables use the **NDB** storage engine; however, an **ndbinfo** entry still appears in the output of `SHOW ENGINES` and `SHOW PLUGINS` as described previously.

You can execute `SELECT` statements against these tables, just as you would normally expect:

```sql
mysql> SELECT * FROM memoryusage;
```

```
<table>
<thead>
<tr>
<th>node_id</th>
<th>memory_type</th>
<th>used</th>
<th>used_pages</th>
<th>total</th>
<th>total_pages</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>Data memory</td>
<td>753664</td>
<td>23</td>
<td>1073741824</td>
<td>32768</td>
</tr>
<tr>
<td>5</td>
<td>Index memory</td>
<td>163840</td>
<td>20</td>
<td>1074003968</td>
<td>131104</td>
</tr>
<tr>
<td>5</td>
<td>Long message buffer</td>
<td>2304</td>
<td>9</td>
<td>67108864</td>
<td>262144</td>
</tr>
</tbody>
</table>
```
The ndbinfo arbitrator_validity_detail Table

The `arbitrator_validity_detail` table shows the view that each data node in the cluster has of the arbitrator. It is a subset of the `membership` table.

The `arbitrator_validity_detail` table contains the following columns:

- **node_id**
  
  This node's node ID

- **arbitrator**
  
  Node ID of arbitrator

More complex queries, such as the two following SELECT statements using the `memoryusage` table, are possible:

```sql
mysql> SELECT SUM(used) as 'Data Memory Used, All Nodes'
FROM memoryusage
WHERE memory_type = 'Data memory';

+---------------------------------------------+
| Data Memory Used, All Nodes                 |
|---------------------------------------------+|
| Data Memory Used, All Nodes                 |
| 6460                                        |
+---------------------------------------------+
1 row in set (0.37 sec)
```

```sql
mysql> SELECT SUM(max) as 'Total IndexMemory Available'
FROM memoryusage
WHERE memory_type = 'Index memory';

+---------------------------------------------+
| Total IndexMemory Available                 |
|---------------------------------------------+|
| Total IndexMemory Available                 |
| 25664                                       |
+---------------------------------------------+
1 row in set (0.33 sec)
```

The `ndbinfo` table and column names are case-sensitive (as is the name of the `ndbinfo` database itself). These identifiers are in lowercase. Trying to use the wrong lettercase results in an error, as shown in this example:

```sql
mysql> SELECT * FROM nodes;

+---------+--------+---------+-------------+
| node_id | uptime | status  | start_phase |
+---------+--------+---------+-------------+
|       1 |  13602 | STARTED |           0 |
|       2 |     16 | STARTED |           0 |
+---------+--------+---------+-------------+
2 rows in set (0.04 sec)
mysql> SELECT * FROM Nodes;
ERROR 1146 (42S02): Table 'ndbinfo.Nodes' doesn't exist
```

`mysqldump` ignores the `ndbinfo` database entirely, and excludes it from any output. This is true even when using the `--databases` or `--all-databases` option.

NDB Cluster also maintains tables in the `INFORMATION_SCHEMA` information database, including the `FILES` table which contains information about files used for NDB Cluster Disk Data storage, and the `ndb_transid_mysql_connection_map` table, which shows the relationships between transactions, transaction coordinators, and NDB Cluster API nodes. For more information, see the descriptions of the tables or Section 7.15, “INFORMATION_SCHEMA Tables for NDB Cluster”.

### 7.14.1 The ndbinfo arbitrator_validity_detail Table

The `arbitrator_validity_detail` table shows the view that each data node in the cluster has of the arbitrator. It is a subset of the `membership` table.

The `arbitrator_validity_detail` table contains the following columns:

- **node_id**
  
  This node's node ID

- **arbitrator**
  
  Node ID of arbitrator
The ndbinfo arbitrator_validity_summary Table

- **arb_ticket**
  Internal identifier used to track arbitration

- **arb_connected**
  Whether this node is connected to the arbitrator; either of Yes or No

- **arb_state**
  Arbitration state

Notes

The node ID is the same as that reported by `ndb_mgm -e "SHOW"`. All nodes should show the same **arbitrator** and **arb_ticket** values as well as the same **arb_state** value. Possible **arb_state** values are ARBIT_NULL, ARBIT_INIT, ARBIT_FIND, ARBIT_PREP1, ARBIT_PREP2, ARBIT_START, ARBIT_RUN, ARBIT_CHOOSE, ARBIT_CRASH, and UNKNOWN.

**arb_connected** shows whether the current node is connected to the arbitrator.

### 7.14.2 The ndbinfo arbitrator_validity_summary Table

The **arbitrator_validity_summary** table provides a composite view of the arbitrator with regard to the cluster's data nodes.

The **arbitrator_validity_summary** table contains the following columns:

- **arbitrator**
  Node ID of arbitrator

- **arb_ticket**
  Internal identifier used to track arbitration

- **arb_connected**
  Whether this arbitrator is connected to the cluster

- **consensus_count**
  Number of data nodes that see this node as arbitrator; either of Yes or No

Notes

In normal operations, this table should have only 1 row for any appreciable length of time. If it has more than 1 row for longer than a few moments, then either not all nodes are connected to the arbitrator, or all nodes are connected, but do not agree on the same arbitrator.

The **arbitrator** column shows the arbitrator's node ID.

**arb_ticket** is the internal identifier used by this arbitrator.

**arb_connected** shows whether this node is connected to the cluster as an arbitrator.

### 7.14.3 The ndbinfo backup_id Table

This table provides a way to find the ID of the backup started most recently for this cluster.
The table contains a single column \texttt{id}, which corresponds to a backup ID taken using the \texttt{ndb_mgm} client \texttt{START BACKUP} command. This table contains a single row.

\textit{Example:} Assume the following sequence of \texttt{START BACKUP} commands issued in the NDB management client, with no other backups taken since the cluster was first started:

```
ndb_mgm> \texttt{START BACKUP}
Waiting for completed, this may take several minutes
Node 5: Backup 1 started from node 50
Node 5: Backup 1 started from node 50 completed
StartGCP: 27894 StopGCP: 27897
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> \texttt{START BACKUP 5}
Waiting for completed, this may take several minutes
Node 5: Backup 5 started from node 50
Node 5: Backup 5 started from node 50 completed
StartGCP: 27905 StopGCP: 27908
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> \texttt{START BACKUP}
Waiting for completed, this may take several minutes
Node 5: Backup 6 started from node 50
Node 5: Backup 6 started from node 50 completed
StartGCP: 27912 StopGCP: 27915
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm> \texttt{START BACKUP 3}
Connected to Management Server at: localhost:1186
Waiting for completed, this may take several minutes
Node 5: Backup 3 started from node 50
Node 5: Backup 3 started from node 50 completed
StartGCP: 28149 StopGCP: 28152
#Records: 2057 #LogRecords: 0
Data: 51580 bytes Log: 0 bytes
ndb_mgm>
```

After this, the \texttt{backup_id} table contains the single row shown here, using the \texttt{mysql} client:

```
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM backup_id;
+----+
<table>
<thead>
<tr>
<th>id</th>
</tr>
</thead>
<tbody>
<tr>
<td>3</td>
</tr>
</tbody>
</table>
+----+
1 row in set (0.00 sec)
```

If no backups can be found, the table contains a single row with 0 as the \texttt{id} value.

The \texttt{backup_id} table was added in NDB 8.0.24.

### 7.14.4 The \texttt{ndbinfo blocks} Table

The \texttt{blocks} table is a static table which simply contains the names and internal IDs of all NDB kernel blocks (see \texttt{NDB Kernel Blocks}). It is for use by the other \texttt{ndbinfo} tables (most of which are actually views) in mapping block numbers to block names for producing human-readable output.

The \texttt{blocks} table contains the following columns:

- \texttt{block_number}
  
  Block number

- \texttt{block_name}
  
  Block name
Notes

To obtain a list of all block names, simply execute:

```
SELECT block_name FROM ndbinfo.blocks;
```

Although this is a static table, its content can vary between different NDB Cluster releases.

7.14.5 The ndbinfo cluster_locks Table

The `cluster_locks` table provides information about current lock requests holding and waiting for locks on NDB tables in an NDB Cluster, and is intended as a companion table to `cluster_operations`. Information obtained from the `cluster_locks` table may be useful in investigating stalls and deadlocks.

The `cluster_locks` table contains the following columns:

- `node_id`:
  ID of reporting node
- `block_instance`:
  ID of reporting LDM instance
- `tableid`:
  ID of table containing this row
- `fragmentid`:
  ID of fragment containing locked row
- `rowid`:
  ID of locked row
- `transid`:
  Transaction ID
- `mode`:
  Lock request mode
- `state`:
  Lock state
- `detail`:
  Whether this is first holding lock in row lock queue
- `op`:
  Operation type
- `duration_millis`:
  Milliseconds spent waiting or holding lock
- `lock_num`:
  ID of lock object
- `waiting_for`:
  Waiting for lock with this ID
The ndbinfo cluster_operations Table

Notes

The table ID (tableid column) is assigned internally, and is the same as that used in other ndbinfo tables. It is also shown in the output of ndb_show_tables.

The transaction ID (transid column) is the identifier generated by the NDB API for the transaction requesting or holding the current lock.

The node column shows the lock mode; this is always one of S (indicating a shared lock) or X (an exclusive lock). If a transaction holds an exclusive lock on a given row, all other locks on that row have the same transaction ID.

The state column shows the lock state. Its value is always one of H (holding) or W (waiting). A waiting lock request waits for a lock held by a different transaction.

When the detail column contains a * (asterisk character), this means that this lock is the first holding lock in the affected row's lock queue; otherwise, this column is empty. This information can be used to help identify the unique entries in a list of lock requests.

The op column shows the type of operation requesting the lock. This is always one of the values READ, INSERT, UPDATE, DELETE, SCAN, or REFRESH.

The duration_millis column shows the number of milliseconds for which this lock request has been waiting or holding the lock. This is reset to 0 when a lock is granted for a waiting request.

The lock ID (lockid column) is unique to this node and block instance.

The lock state is shown in the lock_state column; if this is W, the lock is waiting to be granted, and the waiting_for column shows the lock ID of the lock object this request is waiting for. Otherwise, the waiting_for column is empty. waiting_for can refer only to locks on the same row, as identified by node_id, block_instance, tableid, fragmentid, and rowid.

7.14.6 The ndbinfo cluster_operations Table

The cluster_operations table provides a per-operation (stateful primary key op) view of all activity in the NDB Cluster from the point of view of the local data management (LQH) blocks (see The DBLQH Block).

The cluster_operations table contains the following columns:

- node_id
  Node ID of reporting LQH block
- block_instance
  LQH block instance
- transid
  Transaction ID
- operation_type
  Operation type (see text for possible values)
- state
  Operation state (see text for possible values)
- tableid
  Table ID
The ndbinfo cluster_transactions Table

- **fragmentid**
  Fragment ID
- **client_node_id**
  Client node ID
- **client_block_ref**
  Client block reference
- **tc_node_id**
  Transaction coordinator node ID
- **tc_block_no**
  Transaction coordinator block number
- **tc_block_instance**
  Transaction coordinator block instance

**Notes**

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The `operation_type` column can take any one of the values `READ`, `READ-SH`, `READ-EX`, `INSERT`, `UPDATE`, `DELETE`, `WRITE`, `UNLOCK`, `REFRESH`, `SCAN`, `SCAN-SH`, `SCAN-EX`, or `<unknown>`.

The `state` column can have any one of the values `ABORT_QUEUED`, `ABORT_STOPPED`, `COMMITTED`, `COMMIT_QUEUED`, `COMMIT_STOPPED`, `COPY_CLOSE_STOPPED`, `COPY_FIRST_STOPPED`, `COPY_STOPPED`, `COPY_TUPKEY`, `IDLE`, `LOG_ABORT_QUEUED`, `LOG_ABORT_STOPPED`, `LOG_COMMIT_QUEUED`, `LOG_COMMIT_QUEUED_WAIT_SIGNAL`, `LOG_COMMIT_WRITTEN`, `LOG_COMMIT_WRITTEN_WAIT_SIGNAL`, `LOG_QUEUE`, `PREPARED`, `PREPARED_RECEIVED_COMMIT`, `SCAN_CHECK_STOPPED`, `SCAN_CLOSE_STOPPED`, `SCAN_FIRST_STOPPED`, `SCAN_RELEASE_STOPPED`, `SCAN_STATE_USED`, `SCAN_STOPPED`, `SCAN_TUPKEY`, `STOPPED`, `TC_NOT_CONNECTED`, `WAIT_ACC`, `WAIT_ACC_ABORT`, `WAIT_AI_AFT_ABORT`, `WAIT_ATTR`, `WAIT_SCAN_AI`, `WAIT_TUP`, `WAIT_TUPKEYINFO`, `WAIT_TUP_COMMIT`, or `WAIT_TUP_TO_ABORT`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb$dblqh_tcconnect_state` table, which is normally hidden.)

You can obtain the name of an NDB table from its table ID by checking the output of `ndb_show_tables`.

The `fragid` is the same as the partition number seen in the output of `ndb_desc --extra-partition-info` (short form `-p`).

In `client_node_id` and `client_block_ref`, client refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The `block_instance` and `tc_block_instance` columns provide, respectively, the `DBLQH` and `DBTC` block instance numbers. You can use these along with the block names to obtain information about specific threads from the `threadblocks` table.

### 7.14.7 The ndbinfo cluster_transactions Table

The `cluster_transactions` table shows information about all ongoing transactions in an NDB Cluster.
The cluster_transactions table contains the following columns:

- **node_id**  
  Node ID of transaction coordinator
- **block_instance**  
  TC block instance
- **transid**  
  Transaction ID
- **state**  
  Operation state (see text for possible values)
- **count_operations**  
  Number of stateful primary key operations in transaction (includes reads with locks, as well as DML operations)
- **outstanding_operations**  
  Operations still being executed in local data management blocks
- **inactive_seconds**  
  Time spent waiting for API
- **client_node_id**  
  Client node ID
- **client_block_ref**  
  Client block reference

**Notes**

The transaction ID is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

**block_instance** refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the threadblocks table.

The **state** column can have any one of the values `CS_ABORTING, CS_COMMITTING, CS_COMMIT_SENT, CS_COMPLETE_SENT, CS_COMPLETING, CS_CONNECTED, CS_DISCONNECTED, CS_FAIL_ABORTED, CS_FAIL_ABORTING, CS_FAIL_COMMitted, CS_FAIL_COMMITTING, CS_FAIL_COMPLETED, CS_FAIL_PREPARED, CS_PREPARE_TO_COMMIT, CS_RECEIVING, CS_REC_COMMITTING, CS_RESTART, CS_SEND_FIRE_TRIG_REQ, CS_STARTED, CS_START_COMMITTING, CS_START_SCAN, CS_WAIT_ABORT_CONF, CS_WAIT_COMMIT_CONF, CS_WAIT_COMPLETE_CONF, CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with ndbinfo_show_hidden enabled, you can view this list of states by selecting from the ndb $dbtc_apiconnect_state table, which is normally hidden.)

In **client_node_id** and **client_block_ref**, client refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The **tc_block_instance** column provides the DBTC block instance number. You can use this along with the block name to obtain information about specific threads from the threadblocks table.
7.14.8 The ndbinfo config_nodes Table

The config_nodes table shows nodes configured in an NDB Cluster config.ini file. For each node, the table displays a row containing the node ID, the type of node (management node, data node, or API node), and the name or IP address of the host on which the node is configured to run.

This table does not indicate whether a given node is actually running, or whether it is currently connected to the cluster. Information about nodes connected to an NDB Cluster can be obtained from the nodes and processes table.

The config_nodes table contains the following columns:

- **node_id**
  The node’s ID

- **node_type**
  The type of node

- **node_hostname**
  The name or IP address of the host on which the node resides

Notes

The node_id column shows the node ID used in the config.ini file for this node; if none is specified, the node ID that would be assigned automatically to this node is displayed.

The node_type column displays one of the following three values:

- **MGM**: Management node.
- **NDB**: Data node.
- **API**: API node; this includes SQL nodes.

The node_hostname column shows the node host as specified in the config.ini file. This can be empty for an API node, if HostName has not been set in the cluster configuration file. If HostName has not been set for a data node in the configuration file, localhost is used here. localhost is also used if HostName has not been specified for a management node.

7.14.9 The ndbinfo config_params Table

The config_params table is a static table which provides the names and internal ID numbers of and other information about NDB Cluster configuration parameters. This table can also be used in conjunction with the config_values table for obtaining realtime information about node configuration parameters.

The config_params table contains the following columns:

- **param_number**
  The parameter’s internal ID number

- **param_name**
  The name of the parameter

- **param_description**
  A brief description of the parameter

- **param_type**
The parameter's data type

- **param_default**
  The parameter's default value, if any

- **param_min**
  The parameter's maximum value, if any

- **param_max**
  The parameter's minimum value, if any

- **param_mandatory**
  This is 1 if the parameter is required, otherwise 0

- **param_status**
  Currently unused

**Notes**

This table is read-only.

Although this is a static table, its content can vary between NDB Cluster installations, since supported parameters can vary due to differences between software releases, cluster hardware configurations, and other factors.

### 7.14.10 The ndbinfo config_values Table

The `config_values` table provides information about the current state of node configuration parameter values. Each row in the table corresponds to the current value of a parameter on a given node.

The `config_values` table contains the following columns:

- **node_id**
  ID of the node in the cluster

- **config_param**
  The parameter's internal ID number

- **config_value**
  Current value of the parameter

**Notes**

This table's `config_param` column and the `config_params` table's `param_number` column use the same parameter identifiers. By joining the two tables on these columns, you can obtain detailed information about desired node configuration parameters. The query shown here provides the current values for all parameters on each data node in the cluster, ordered by node ID and parameter name:

```sql
SELECT v.node_id AS 'Node Id',
       p.param_name AS 'Parameter',
       v.config_value AS 'Value'
FROM config_values v
JOIN config_params p
ON v.config_param=p.param_number
WHERE p.param_name NOT LIKE '\\_%'
ORDER BY v.node_id, p.param_name;
```
Partial output from the previous query when run on a small example cluster used for simple testing:

<table>
<thead>
<tr>
<th>Node Id</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>Arbitration</td>
<td>1</td>
</tr>
<tr>
<td>2</td>
<td>ArbitrationTimeout</td>
<td>7500</td>
</tr>
<tr>
<td>2</td>
<td>BackupDataBufferSize</td>
<td>16777216</td>
</tr>
<tr>
<td>2</td>
<td>BackupDataDir</td>
<td>/home/jon/data</td>
</tr>
<tr>
<td>2</td>
<td>BackupDiskWriteSpeedPct</td>
<td>50</td>
</tr>
<tr>
<td>2</td>
<td>BackupLogBufferSize</td>
<td>16777216</td>
</tr>
<tr>
<td>...</td>
<td>TotalSendBufferMemory</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>TransactionBufferMemory</td>
<td>1048576</td>
</tr>
<tr>
<td>3</td>
<td>TransactionDeadlockDetectionTimeout</td>
<td>1200</td>
</tr>
<tr>
<td>3</td>
<td>TransactionInactiveTimeout</td>
<td>4294967039</td>
</tr>
<tr>
<td>3</td>
<td>TwoPassInitialNodeRestartCopy</td>
<td>0</td>
</tr>
<tr>
<td>3</td>
<td>UndoDataBuffer</td>
<td>16777216</td>
</tr>
<tr>
<td>3</td>
<td>UndoIndexBuffer</td>
<td>2097152</td>
</tr>
</tbody>
</table>

248 rows in set (0.02 sec)

The WHERE clause filters out parameters whose names begin with a double underscore (__); these parameters are reserved for testing and other internal uses by the NDB developers, and are not intended for use in a production NDB Cluster.

You can obtain output that is more specific, more detailed, or both by issuing the proper queries. This example provides all types of available information about the NodeId, NoOfReplicas, HostName, DataMemory, IndexMemory, and TotalSendBufferMemory parameters as currently set for all data nodes in the cluster:

```sql
SELECT p.param_name AS Name, v.node_id AS Node, p.param_type AS Type, p.param_default AS 'Default', p.param_min AS Minimum, p.param_max AS Maximum, CASE p.param_mandatory WHEN 1 THEN 'Y' ELSE 'N' END AS 'Required', v.config_value AS Current
FROM config_params p
JOIN config_values v
ON p.param_number = v.config_param
WHERE p.param_name IN ('NodeId', 'NoOfReplicas', 'HostName', 'DataMemory', 'IndexMemory', 'TotalSendBufferMemory')
```

The output from this query when run on a small NDB Cluster with 2 data nodes used for simple testing is shown here (NDB 8.0.18 and later):

```
*************** 1. row ***************
Name: NodeId
Node: 2
Type: unsigned
Default: 1
Minimum: 1
Maximum: 144
Required: Y
Current: 2

*************** 2. row ***************
Name: HostName
Node: 2
Type: string
Default: localhost
Minimum: 
Maximum: 
Required: N
Current: 127.0.0.1

*************** 3. row ***************
Name: TotalSendBufferMemory
Node: 2
```
<table>
<thead>
<tr>
<th>Name</th>
<th>Node</th>
<th>Type</th>
<th>Default</th>
<th>Minimum</th>
<th>Maximum</th>
<th>Required</th>
<th>Current</th>
</tr>
</thead>
<tbody>
<tr>
<td>NoOfReplicas</td>
<td>2</td>
<td>unsigned</td>
<td>2</td>
<td>1</td>
<td>4</td>
<td>N</td>
<td>2</td>
</tr>
<tr>
<td>DataMemory</td>
<td>2</td>
<td>unsigned</td>
<td>102760448</td>
<td>1048576</td>
<td>1099511627776</td>
<td>N</td>
<td>524288000</td>
</tr>
<tr>
<td>NodeId</td>
<td>3</td>
<td>unsigned</td>
<td></td>
<td>1</td>
<td>144</td>
<td>Y</td>
<td>3</td>
</tr>
<tr>
<td>HostName</td>
<td>3</td>
<td>string</td>
<td>localhost</td>
<td></td>
<td></td>
<td>N</td>
<td>127.0.0.1</td>
</tr>
<tr>
<td>TotalSendBufferMemory</td>
<td>3</td>
<td>unsigned</td>
<td>0</td>
<td>262144</td>
<td>4294967039</td>
<td>N</td>
<td>0</td>
</tr>
</tbody>
</table>

10 rows in set (0.01 sec)

7.14.11 The ndbinfo counters Table
The counters table provides running totals of events such as reads and writes for specific kernel blocks and data nodes. Counts are kept from the most recent node start or restart; a node start or restart resets all counters on that node. Not all kernel blocks have all types of counters.

The counters table contains the following columns:

- **node_id**
  The data node ID

- **block_name**
  Name of the associated NDB kernel block (see NDB Kernel Blocks).

- **block_instance**
  Block instance

- **counter_id**
  The counter's internal ID number; normally an integer between 1 and 10, inclusive.

- **counter_name**
  The name of the counter. See text for names of individual counters and the NDB kernel block with which each counter is associated.

- **val**
  The counter's value

**Notes**

Each counter is associated with a particular NDB kernel block.

The **OPERATIONS** counter is associated with the **DBLQH** (local query handler) kernel block. A primary-key read counts as one operation, as does a primary-key update. For reads, there is one operation in **DBLQH** per operation in **DBTC**. For writes, there is one operation counted per fragment replica.


**LOCAL_WRITES** and **LOCAL_READS** are primary-key operations using a transaction coordinator in a node that also holds the primary fragment replica of the record.

The **READS** counter includes all reads. **LOCAL_READS** includes only those reads of the primary fragment replica on the same node as this transaction coordinator. **SIMPLE_READS** includes only those reads in which the read operation is the beginning and ending operation for a given transaction. Simple reads do not hold locks but are part of a transaction, in that they observe uncommitted changes made by the transaction containing them but not of any other uncommitted transactions. Such reads are “simple” from the point of view of the TC block; since they hold no locks they are not durable, and once **DBTC** has routed them to the relevant LQH block, it holds no state for them.

**ATTRINFO** keeps a count of the number of times an interpreted program is sent to the data node. See **NDB Protocol Messages**, for more information about **ATTRINFO** messages in the NDB kernel.

The block_name and block_instance columns provide, respectively, the applicable NDB kernel block name and instance number. You can use these to obtain information about specific threads from the threadblocks table.

A number of counters provide information about transporter overload and send buffer sizing when troubleshooting such issues. For each LQH instance, there is one instance of each counter in the following list:

- **LQKEY_OVERLOAD**: Number of primary key requests rejected at the LQH block instance due to transporter overload
- **LQKEY_OVERLOAD_TC**: Count of instances of LQKEY_OVERLOAD where the TC node transporter was overloaded
- **LQKEY_OVERLOAD_READER**: Count of instances of LQKEY_OVERLOAD where the API reader (reads only) node was overloaded.
- **LQKEY_OVERLOAD_NODE_PEER**: Count of instances of LQKEY_OVERLOAD where the next backup data node (writes only) was overloaded.
- **LQKEY_OVERLOAD_SUBSCRIBER**: Count of instances of LQKEY_OVERLOAD where a event subscriber (writes only) was overloaded.
- **LQSCAN_SLOWDOWNS**: Count of instances where a fragment scan batch size was reduced due to scanning API transporter overload.

### 7.14.12 The ndbinfo cpudata Table

The cpudata table provides data about CPU usage during the last second.

The cpustat table contains the following columns:

- **node_id**
  Node ID
- **cpu_no**
  CPU ID
- **cpu_online**
  1 if the CPU is currently online, otherwise 0
- **cpu_userspace_time**
  CPU time spent in userspace
- **cpu_idle_time**
  CPU time spent idle
- **cpu_system_time**
  CPU time spent in system time
- **cpu_interrupt_time**
  CPU time spent handling interrupts (hardware and software)
- **cpu_exec_vm_time**
  CPU time spent in virtual machine execution
The `cpudata` table is available only on Linux and Solaris operating systems.

This table was added in NDB 8.0.23.

### 7.14.13 The ndbinfo `cpudata_1sec` Table

The `cpudata_1sec` table provides data about CPU usage per second over the last 20 seconds.

The `cpustat` table contains the following columns:

- `node_id`
  
  Node ID
  
- `measurement_id`
  
  Measurement sequence ID; later measurements have lower IDs
  
- `cpu_no`
  
  CPU ID
  
- `cpu_online`
  
  1 if the CPU is currently online, otherwise 0
  
- `cpu_userspace_time`
  
  CPU time spent in userspace
  
- `cpu_idle_time`
  
  CPU time spent idle
  
- `cpu_system_time`
  
  CPU time spent in system time
  
- `cpu_interrupt_time`
  
  CPU time spent handling interrupts (hardware and software)
  
- `cpu_exec_vm_time`
  
  CPU time spent in virtual machine execution
  
- `elapsed_time`
  
  Time in microseconds used for this measurement

### Notes

The `cpudata_1sec` table is available only on Linux and Solaris operating systems.

This table was added in NDB 8.0.23.

### 7.14.14 The ndbinfo `cpudata_20sec` Table

The `cpudata_20sec` table provides data about CPU usage per 20-second interval over the last 400 seconds.
The `cpustat` table contains the following columns:

- `node_id`
  Node ID
- `measurement_id`
  Measurement sequence ID; later measurements have lower IDs
- `cpu_no`
  CPU ID
- `cpu_online`
  1 if the CPU is currently online, otherwise 0
- `cpu_userspace_time`
  CPU time spent in userspace
- `cpu_idle_time`
  CPU time spent idle
- `cpu_system_time`
  CPU time spent in system time
- `cpu_interrupt_time`
  CPU time spent handling interrupts (hardware and software)
- `cpu_exec_vm_time`
  CPU time spent in virtual machine execution
- `elapsed_time`
  Time in microseconds used for this measurement

Notes

The `cpudata_20sec` table is available only on Linux and Solaris operating systems.

This table was added in NDB 8.0.23.

### 7.14.15 The `ndbinfo cpudata_50ms` Table

The `cpudata_50ms` table provides data about CPU usage per 50-millisecond interval over the last second.

The `cpustat` table contains the following columns:

- `node_id`
  Node ID
- `measurement_id`
  Measurement sequence ID; later measurements have lower IDs
The `ndbinfo cpuinfo` Table

- **cpu_no**
  
  CPU ID

- **cpu_online**
  
  1 if the CPU is currently online, otherwise 0

- **cpu_userspace_time**
  
  CPU time spent in userspace

- **cpu_idle_time**
  
  CPU time spent idle

- **cpu_system_time**
  
  CPU time spent in system time

- **cpu_interrupt_time**
  
  CPU time spent handling interrupts (hardware and software)

- **cpu_exec_vm_time**
  
  CPU time spent in virtual machine execution

- **elapsed_time**
  
  Time in microseconds used for this measurement

**Notes**

The `cpudata_50ms` table is available only on Linux and Solaris operating systems.

This table was added in NDB 8.0.23.

### 7.14.16 The `ndbinfo cpuinfo` Table

The `cpuinfo` table provides information about the CPU on which a given data node executes.

The `cpuinfo` table contains the following columns:

- **node_id**
  
  Node ID

- **cpu_no**
  
  CPU ID

- **cpu_online**
  
  1 if the CPU is online, otherwise 0

- **core_id**
  
  CPU core ID

- **socket_id**
  
  CPU socket ID
Notes

The cpuinfo table is available on all operating systems supported by NDB, with the exception of MacOS and FreeBSD.

This table was added in NDB 8.0.23.

7.14.17 The ndbinfo cpustat Table

The cpustat table provides per-thread CPU statistics gathered each second, for each thread running in the NDB kernel.

The cpustat table contains the following columns:

- **node_id**
  ID of the node where the thread is running
- **thr_no**
  Thread ID (specific to this node)
- **OS_user**
  OS user time
- **OS_system**
  OS system time
- **OS_idle**
  OS idle time
- **thread_exec**
  Thread execution time
- **thread_sleeping**
  Thread sleep time
- **thread_spinning**
  Thread spin time
- **thread_send**
  Thread send time
- **thread_buffer_full**
  Thread buffer full time
- **elapsed_time**
  Elapsed time

7.14.18 The ndbinfo cpustat_50ms Table

The cpustat_50ms table provides raw, per-thread CPU data obtained each 50 milliseconds for each thread running in the NDB kernel.
Like `cpustat_1sec` and `cpustat_20sec`, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, `cpsustat_50ms` provides 1 second of history.

The `cpustat_50ms` table contains the following columns:

- **node_id**
  ID of the node where the thread is running
- **thr_no**
  Thread ID (specific to this node)
- **OS_user_time**
  OS user time
- **OS_system_time**
  OS system time
- **OS_idle_time**
  OS idle time
- **exec_time**
  Thread execution time
- **sleep_time**
  Thread sleep time
- **spin_time**
  Thread spin time
- **send_time**
  Thread send time
- **buffer_full_time**
  Thread buffer full time
- **elapsed_time**
  Elapsed time

### 7.14.19 The ndbinfo cpustat_1sec Table

The `cpustat-1sec` table provides raw, per-thread CPU data obtained each second for each thread running in the NDB kernel.

Like `cpustat_50ms` and `cpustat_20sec`, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, `cpsustat_1sec` provides 20 seconds of history.

The `cpustat_1sec` table contains the following columns:

- **node_id**
  ID of the node where the thread is running
- **thr_no**
The ndbinfo cpustat_20sec Table

The cpustat_20sec table provides raw, per-thread CPU data obtained each 20 seconds, for each thread running in the NDB kernel.

Like cpustat_50ms and cpustat_1sec, this table shows 20 measurement sets per thread, each referencing a period of the named duration. Thus, cpustat_20sec provides 400 seconds of history.

The cpustat_20sec table contains the following columns:

- node_id
  ID of the node where the thread is running

- thr_no
  Thread ID (specific to this node)

- OS_user_time
  OS user time

- OS_system_time
  OS system time
The ndbinfo dict_obj_info Table

- **OS_idle_time**
  OS idle time
- **exec_time**
  Thread execution time
- **sleep_time**
  Thread sleep time
- **spin_time**
  Thread spin time
- **send_time**
  Thread send time
- **buffer_full_time**
  Thread buffer full time
- **elapsed_time**
  Elapsed time

### 7.14.21 The ndbinfo dict_obj_info Table

The **dict_obj_info** table provides information about NDB data dictionary (DICT) objects such as tables and indexes. (The **dict_obj_types** table can be queried for a list of all the types.) This information includes the object's type, state, parent object (if any), and fully qualified name.

The **dict_obj_info** table contains the following columns:

- **type**
  Type of DICT object; join on **dict_obj_types** to obtain the name
- **id**
  Object identifier; for Disk Data undo log files and data files, this is the same as the value shown in the **LOGFILE_GROUP_NUMBER** column of the **INFORMATION_SCHEMA.FILES** table; for undo log files, it is also the same as the value shown for the **log_id** column in the **ndbinfo logbuffers** and **logspaces** tables
- **version**
  Object version
- **state**
  Object state
- **parent_obj_type**
  Parent object's type (a **dict_obj_types** type ID); 0 indicates that the object has no parent
- **parent_obj_id**
  Parent object ID (such as a base table); 0 indicates that the object has no parent
- **fq_name**
7.14.22 The ndbinfo dict_obj_tree Table

The dict_obj_tree table provides a tree-based view of table information from the dict_obj_info table. This is intended primarily for use in testing, but can be useful in visualizing hierarchies of NDB database objects.

The dict_obj_tree table contains the following columns:

- **type**
  Type of DICT object; join on dict_obj_types to obtain the name of the object type

- **id**
  Object identifier; same as the id column in dict_obj_info

  For Disk Data undo log files and data files, this is the same as the value shown in the LOGFILE_GROUP_NUMBER column of the INFORMATION_SCHEMA.FILES table; for undo log files, it also the same as the value shown for the log_id column in the ndbinfo logbuffers and logspaces tables

- **name**
  The fully qualified name of the object; the same as the fq_name column in dict_obj_info

  For a table, this is database_name/def/table_name (the same as its parent_name); for an index of any type, this takes the form NDB$INDEX_index_id_CUSTOM

- **parent_type**
  The DICT object type of this object's parent object; join on dict_obj_types to obtain the name of the object type

- **parent_id**
  Identifier for this object's parent object; the same as the dict_obj_info table's id column

- **parent_name**
  Fully qualified name of this object's parent object; the same as the dict_obj_info table's fq_name column

  For a table, this has the form database_name/def/table_name. For an index, the name is sys/def/table_id/index_name. For a primary key, it is sys/def/table_id/PRIMARY, and for a unique key it is sys/def/table_id/uk_name$unique

- **root_type**
  The DICT object type of the root object; join on dict_obj_types to obtain the name of the object type

- **root_id**
  Identifier for the root object; the same as the dict_obj_info table's id column

- **root_name**
  Fully qualified name of the root object; the same as the dict_obj_info table's fq_name column
The ndbinfo dict_obj_tree Table

- **level**
  Level of the object in the hierarchy

- **path**
  Complete path to the object in the NDB object hierarchy; objects are separated by a right arrow (represented as `->`), starting with the root object on the left

- **indented_name**
  The name prefixed with a right arrow (represented as `->`) with a number of spaces preceding it that correspond to the object's depth in the hierarchy

The *path* column is useful for obtaining a complete path to a given NDB database object in a single line, whereas the *indented_name* column can be used to obtain a tree-like layout of complete hierarchy information for a desired object.

**Example:** Assuming the existence of a `test` database and no existing table named `t1` in this database, execute the following SQL statement:

```sql
CREATE TABLE test.t1 (
  a INT PRIMARY KEY,
  b INT,
  UNIQUE KEY(b)
) ENGINE = NDB;
```

You can obtain the path to the table just created using the query shown here:

```sql
mysql> SELECT path FROM ndbinfo.dict_obj_tree
  -> WHERE name LIKE 'test%t1';
```

```plaintext
+-------------+
| path        |
+-------------+
| test/def/t1 |
+-------------+
1 row in set (0.14 sec)
```

You can see the paths to all dependent objects of this table using the path to the table as the root name in a query like this one:

```sql
mysql> SELECT path FROM ndbinfo.dict_obj_tree
  -> WHERE root_name = 'test/def/t1';
```

```plaintext
+------------------------------------------------------------------+
| path                                                                |
+------------------------------------------------------------------+
| test/def/t1                                                         |
| test/def/t1 -> sys/def/13/b                                         |
| test/def/t1 -> sys/def/13/b -> NDB$INDEX_15_CUSTOM                 |
| test/def/t1 -> sys/def/13/b$unique -> NDB$INDEX_16_UI              |
| test/def/t1 -> sys/def/13/PRIMARY -> NDB$INDEX_14_CUSTOM           |
+------------------------------------------------------------------+
7 rows in set (0.16 sec)
```

To obtain a hierarchical view of the `t1` table with all its dependent objects, execute a query similar to this one which selects the indented name of each object having `test/def/t1` as the name of its root object:

```sql
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
  -> WHERE root_name = 'test/def/t1';
```

```plaintext
+----------------------------+
| indented_name              |
+----------------------------+
| test/def/t1                |
|   -> sys/def/13/b          |
+----------------------------+
```
When working with Disk Data tables, note that, in this context, a tablespace or log file group is considered a root object. This means that you must know the name of any tablespace or log file group associated with a given table, or obtain this information from `SHOW CREATE TABLE` and then querying `INFORMATION_SCHEMA.FILES`, or similar means as shown here:

```sql
mysql> SHOW CREATE TABLE test.dt_1;
```

```
Table: dt_1
```

```
CREATE TABLE `dt_1` (
  `member_id` int unsigned NOT NULL AUTO_INCREMENT,
  `last_name` varchar(50) NOT NULL,
  `first_name` varchar(50) NOT NULL,
  `dob` date NOT NULL,
  `joined` date NOT NULL,
  PRIMARY KEY (`member_id`),
  KEY `last_name` (`last_name`,`first_name`)
) CHARACTER SET utf8mb4 COLLATE utf8mb4_0900_ai_ci
```

Now you can obtain hierarchical information for the table, tablespace, and log file group like this:

```sql
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
WHERE root_name = 'test/def/dt_1';
```

```
<table>
<thead>
<tr>
<th>indented_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>test/def/dt_1</td>
</tr>
<tr>
<td>-&gt; sys/def/23/last_name</td>
</tr>
<tr>
<td>-&gt; NDB$INDEX_25_CUSTOM</td>
</tr>
<tr>
<td>-&gt; sys/def/23/PRIMARY</td>
</tr>
<tr>
<td>-&gt; NDB$INDEX_24_CUSTOM</td>
</tr>
</tbody>
</table>
```

```sql
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
WHERE root_name = 'ts_1';
```

```
<table>
<thead>
<tr>
<th>indented_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>ts_1</td>
</tr>
<tr>
<td>-&gt; data_1.dat</td>
</tr>
<tr>
<td>-&gt; data_2.dat</td>
</tr>
</tbody>
</table>
```

```sql
mysql> SELECT indented_name FROM ndbinfo.dict_obj_tree
WHERE root_name LIKE 'lg_1';
```

```
<table>
<thead>
<tr>
<th>indented_name</th>
</tr>
</thead>
<tbody>
<tr>
<td>lg_1</td>
</tr>
<tr>
<td>-&gt; undo_1.log</td>
</tr>
<tr>
<td>-&gt; undo_2.log</td>
</tr>
</tbody>
</table>
```

The `dict_obj_tree` table was added in NDB 8.0.24.
The ndbinfo dict_obj_types Table

7.14.23 The ndbinfo dict_obj_types Table

The `dict_obj_types` table is a static table listing possible dictionary object types used in the NDB kernel. These are the same types defined by `Object::Type` in the NDB API.

The `dict_obj_types` table contains the following columns:

- **type_id**
  The type ID for this type

- **type_name**
  The name of this type

7.14.24 The ndbinfo disk_write_speed_base Table

The `disk_write_speed_base` table provides base information about the speed of disk writes during LCP, backup, and restore operations.

The `disk_write_speed_base` table contains the following columns:

- **node_id**
  Node ID of this node

- **thr_no**
  Thread ID of this LDM thread

- **millis_ago**
  Milliseconds since this reporting period ended

- **millis_passed**
  Milliseconds elapsed in this reporting period

- **backup_lcp_bytes_written**
  Number of bytes written to disk by local checkpoints and backup processes during this period

- **redo_bytes_written**
  Number of bytes written to REDO log during this period

- **target_disk_write_speed**
  Actual speed of disk writes per LDM thread (base data)

7.14.25 The ndbinfo disk_write_speed_aggregate Table

The `disk_write_speed_aggregate` table provides aggregated information about the speed of disk writes during LCP, backup, and restore operations.

The `disk_write_speed_aggregate` table contains the following columns:

- **node_id**
  Node ID of this node

- **thr_no**
Thread ID of this LDM thread

- backup_lcp_speed_last_sec
  Number of bytes written to disk by backup and LCP processes in the last second

- redo_speed_last_sec
  Number of bytes written to REDO log in the last second

- backup_lcp_speed_last_10sec
  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- redo_speed_last_10sec
  Number of bytes written to REDO log per second, averaged over the last 10 seconds

- std_dev_backup_lcp_speed_last_10sec
  Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

- std_dev_redo_speed_last_10sec
  Standard deviation in number of bytes written to REDO log per second, averaged over the last 10 seconds

- backup_lcp_speed_last_60sec
  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- redo_speed_last_60sec
  Number of bytes written to REDO log per second, averaged over the last 10 seconds

- std_dev_backup_lcp_speed_last_60sec
  Standard deviation in number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

- std_dev_redo_speed_last_60sec
  Standard deviation in number of bytes written to REDO log per second, averaged over the last 60 seconds

- slowdowns_due_to_io_lag
  Number of seconds since last node start that disk writes were slowed due to REDO log I/O lag

- slowdowns_due_to_high_cpu
  Number of seconds since last node start that disk writes were slowed due to high CPU usage

- disk_write_speed_set_to_min
  Number of seconds since last node start that disk write speed was set to minimum

- current_target_disk_write_speed
  Actual speed of disk writes per LDM thread (aggregated)
The ndbinfo disk_write_speed_aggregate_node Table

7.14.26 The ndbinfo disk_write_speed_aggregate_node Table

The disk_write_speed_aggregate_node table provides aggregated information per node about the speed of disk writes during LCP, backup, and restore operations.

The disk_write_speed_aggregate_node table contains the following columns:

• node_id
  Node ID of this node

• backup_lcp_speed_last_sec
  Number of bytes written to disk by backup and LCP processes in the last second

• redo_speed_last_sec
  Number of bytes written to REDO log in the last second

• backup_lcp_speed_last_10sec
  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 10 seconds

• redo_speed_last_10sec
  Number of bytes written to REDO log per second, averaged over the last 10 seconds

• backup_lcp_speed_last_60sec
  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

• redo_speed_last_60sec
  Number of bytes written to disk by backup and LCP processes per second, averaged over the last 60 seconds

7.14.27 The ndbinfo diskpagebuffer Table

The diskpagebuffer table provides statistics about disk page buffer usage by NDB Cluster Disk Data tables.

The diskpagebuffer table contains the following columns:

• node_id
  The data node ID

• block_instance
  Block instance

• pages_written
  Number of pages written to disk.

• pages_written_lcp
  Number of pages written by local checkpoints.

• pages_read
The ndbinfo diskpagebuffer Table

Number of pages read from disk

- `log_waits`
  Number of page writes waiting for log to be written to disk

- `page_requests_direct_return`
  Number of requests for pages that were available in buffer

- `page_requests_wait_queue`
  Number of requests that had to wait for pages to become available in buffer

- `page_requests_wait_io`
  Number of requests that had to be read from pages on disk (pages were unavailable in buffer)

Notes

You can use this table with NDB Cluster Disk Data tables to determine whether `DiskPageBufferMemory` is sufficiently large to allow data to be read from the buffer rather than from disk; minimizing disk seeks can help improve performance of such tables.

You can determine the proportion of reads from `DiskPageBufferMemory` to the total number of reads using a query such as this one, which obtains this ratio as a percentage:

```sql
SELECT
  node_id,
  100 * page_requests_direct_return / (page_requests_direct_return + page_requests_wait_io) AS hit_ratio
FROM ndbinfo.diskpagebuffer;
```

The result from this query should be similar to what is shown here, with one row for each data node in the cluster (in this example, the cluster has 4 data nodes):

<table>
<thead>
<tr>
<th>node_id</th>
<th>hit_ratio</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>97.6744</td>
</tr>
<tr>
<td>6</td>
<td>97.6879</td>
</tr>
<tr>
<td>7</td>
<td>98.1776</td>
</tr>
<tr>
<td>8</td>
<td>98.1343</td>
</tr>
</tbody>
</table>

The `hit_ratio` values approaching 100% indicate that only a very small number of reads are being made from disk rather than from the buffer, which means that Disk Data read performance is approaching an optimum level. If any of these values are less than 95%, this is a strong indicator that the setting for `DiskPageBufferMemory` needs to be increased in the `config.ini` file.

Note

A change in `DiskPageBufferMemory` requires a rolling restart of all of the cluster's data nodes before it takes effect.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table. Using this information, you can obtain information about disk page buffer metrics relating to individual threads; an example query using `LIMIT 1` to limit the output to a single thread is shown here:

```
mysql> SELECT
```
The ndbinfo diskstat Table

The `diskstat` table provides information about writes to Disk Data tablespaces during the past 1 second.

The `diskstat` table contains the following columns:

- **node_id**
  Node ID of this node

- **block_instance**
  ID of reporting instance of PGMAN

- **pages_made_dirty**
  Number of pages made dirty during the past second

- **reads_issued**
  Reads issued during the past second

- **reads_completed**
  Reads completed during the past second

- **writes_issued**
  Writes issued during the past second

- **writes_completed**
  Writes completed during the past second

- **log_writes_issued**
  Number of times a page write has required a log write during the past second

- **log_writes_completed**
  Number of log writes completed during the last second

- **get_page_calls_issued**

```sql
> node_id, thr_no, block_name, thread_name, pages_written,
> pages_written_lcp, pages_read, log_waits,
> page_requests_direct_return, page_requests_wait_queue,
> page_requests_wait_io
> FROM ndbinfo.diskpagebuffer
> INNER JOIN ndbinfo.threadblocks USING (node_id, block_instance)
> INNER JOIN ndbinfo.threads USING (node_id, thr_no)
> WHERE block_name = 'PGMAN' LIMIT 1\G

*************************** 1. row ***************************
  node_id: 1
  thr_no: 1
  block_name: PGMAN
  thread_name: rep
  pages_written: 0
  pages_written_lcp: 0
  pages_read: 1
  log_waits: 0
  page_requests_direct_return: 4
  page_requests_wait_queue: 0
  page_requests_wait_io: 1
1 row in set (0.01 sec)
```
The ndbinfo diskstats_1sec Table

- **get_page()**
  - **get_page_reqs_issued**
    - Number of times that a **get_page()** call has resulted in a wait for I/O or completion of I/O already begun during the past second
  - **get_page_reqs_completed**
    - Number of **get_page()** calls waiting for I/O or I/O completion that have completed during the past second

**Notes**

Each row in this table corresponds to an instance of **PGMAN**; there is one such instance per LDM thread plus an additional instance for each data node.

The **diskstat** table was added in NDB 8.0.19.

### 7.14.29 The ndbinfo diskstats_1sec Table

The **diskstats_1sec** table provides information about writes to Disk Data tablespaces over the past 20 seconds.

The **diskstat** table contains the following columns:

- **node_id**
  - Node ID of this node
- **block_instance**
  - ID of reporting instance of **PGMAN**
- **pages_made_dirty**
  - Pages made dirty during the designated 1-second interval
- **reads_issued**
  - Reads issued during the designated 1-second interval
- **reads_completed**
  - Reads completed during the designated 1-second interval
- **writes_issued**
  - Writes issued during the designated 1-second interval
- **writes_completed**
  - Writes completed during the designated 1-second interval
- **log_writes_issued**
  - Number of times a page write has required a log write during the designated 1-second interval
- **log_writes_completed**
  - Number of log writes completed during the designated 1-second interval
The ndbinfo error_messages Table

• **get_page_calls_issued**
  Number of **get_page()** calls issued during the designated 1-second interval

• **get_page_reqs_issued**
  Number of times that a **get_page()** call has resulted in a wait for I/O or completion of I/O already begun during the designated 1-second interval

• **get_page_reqs_completed**
  Number of **get_page()** calls waiting for I/O or I/O completion that have completed during the designated 1-second interval

• **seconds_ago**
  Number of 1-second intervals in the past of the interval to which this row applies

**Notes**

Each row in this table corresponds to an instance of **PGMAN** during a 1-second interval occurring from 0 to 19 seconds ago; there is one such instance per LDM thread plus an additional instance for each data node.

The **diskstats_1sec** table was added in NDB 8.0.19.

7.14.30 The ndbinfo error_messages Table

The **error_messages** table provides information about

The **error_messages** table contains the following columns:

• **error_code**
  Numeric error code

• **error_description**
  Description of error

• **error_status**
  Error status code

• **error_classification**
  Error classification code

**Notes**

**error_code** is a numeric NDB error code. This is the same error code that can be supplied to **ndb_perror** (or, prior to NDB 8.0.13, to **perror --ndb**).

**error_description** provides a basic description of the condition causing the error.

The **error_status** column provides status information relating to the error. Possible values for this column are listed here:

• No error
• Illegal connect string
The ndbinfo hwinfo Table

- Illegal server handle
- Illegal reply from server
- Illegal number of nodes
- Illegal node status
- Out of memory
- Management server not connected
- Could not connect to socket
- Start failed
- Stop failed
- Restart failed
- Could not start backup
- Could not abort backup
- Could not enter single user mode
- Could not exit single user mode
- Failed to complete configuration change
- Failed to get configuration
- Usage error
- Success
- Permanent error
- Temporary error
- Unknown result
- Temporary error, restart node
- Permanent error, external action needed
- Ndbd file system error, restart node initial
- Unknown

The error_classification column shows the error classification. See NDB Error Classifications, for information about classification codes and their meanings.

7.14.31 The ndbinfo hwinfo Table

The hwinfo table provides information about the hardware on which a given data node executes.

The hwinfo table contains the following columns:

- node_id
  Node ID
- cpu_cnt_max
The ndbinfo locks_per_fragment Table

Number of processors on this host
• cpu_cnt

Number of processors available to this node
• num_cpu_cores

Number of CPU cores on this host
• num_cpu_sockets

Number of CPU sockets on this host
• HW_memory_size

Amount of memory available on this host
• model_name

CPU model name

Notes
The hwinf0 table is available on all operating systems supported by NDB.
This table was added in NDB 8.0.23.

7.14.32 The ndbinfo locks_per_fragment Table

The locks_per_fragment table provides information about counts of lock claim requests, and the outcomes of these requests on a per-fragment basis, serving as a companion table to operations_per_fragment and memory_per_fragment. This table also shows the total time spent waiting for locks successfully and unsuccessfully since fragment or table creation, or since the most recent restart.

The locks_per_fragment table contains the following columns:

• fq_name
  Fully qualified table name

• parent fq_name
  Fully qualified name of parent object

• type
  Table type; see text for possible values

• table_id
  Table ID

• node_id
  Reporting node ID

• block_instance
  LDM instance ID
The ndbinfo locks_per_fragment Table

- **fragment_num**
  Fragment identifier
- **ex_req**
  Exclusive lock requests started
- **ex_imm_ok**
  Exclusive lock requests immediately granted
- **ex_wait_ok**
  Exclusive lock requests granted following wait
- **ex_wait_fail**
  Exclusive lock requests not granted
- **sh_req**
  Shared lock requests started
- **sh_imm_ok**
  Shared lock requests immediately granted
- **sh_wait_ok**
  Shared lock requests granted following wait
- **sh_wait_fail**
  Shared lock requests not granted
- **wait_ok_millis**
  Time spent waiting for lock requests that were granted, in milliseconds
- **wait_fail_millis**
  Time spent waiting for lock requests that failed, in milliseconds

**Notes**

- **block_instance** refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

- **fq_name** is a fully qualified database object name in `database/schema/name` format, such as `test/def/t1` or `sys/def/10/b$unique`.

- **parent_fq_name** is the fully qualified name of this object's parent object (table).

- **table_id** is the table's internal ID generated by NDB. This is the same internal table ID shown in other ndbinfo tables; it is also visible in the output of `ndb_show_tables`.

- The **type** column shows the type of table. This is always one of System table, User table, Unique hash index, Hash index, Unique ordered index, Ordered index, Hash index trigger, Subscription trigger, Read only constraint, Index trigger, Reorganize trigger, Tablespace, Log file group, Data file, Undo file, Hash map, Foreign key definition, Foreign key parent trigger, Foreign key child trigger, or Schema transaction.
The ndbinfo logbuffers Table

The values shown in all of the columns \( \text{ex\_req, ex\_req\_imm\_ok, ex\_wait\_ok, ex\_wait\_fail, sh\_req, sh\_req\_imm\_ok, sh\_wait\_ok, and sh\_wait\_fail} \) represent cumulative numbers of requests since the table or fragment was created, or since the last restart of this node, whichever of these occurred later. This is also true for the time values shown in the \( \text{wait\_ok\_millis and wait\_fail\_millis} \) columns.

Every lock request is considered either to be in progress, or to have completed in some way (that is, to have succeeded or failed). This means that the following relationships are true:

\[
\text{ex\_req} \geq (\text{ex\_req\_imm\_ok} + \text{ex\_wait\_ok} + \text{ex\_wait\_fail})
\]
\[
\text{sh\_req} \geq (\text{sh\_req\_imm\_ok} + \text{sh\_wait\_ok} + \text{sh\_wait\_fail})
\]

The number of requests currently in progress is the current number of incomplete requests, which can be found as shown here:

\[
[\text{exclusive lock requests in progress}] = \text{ex\_req} - (\text{ex\_req\_imm\_ok} + \text{ex\_wait\_ok} + \text{ex\_wait\_fail})
\]
\[
[\text{shared lock requests in progress}] = \text{sh\_req} - (\text{sh\_req\_imm\_ok} + \text{sh\_wait\_ok} + \text{sh\_wait\_fail})
\]

A failed wait indicates an aborted transaction, but the abort may or may not be caused by a lock wait timeout. You can obtain the total number of aborts while waiting for locks as shown here:

\[
[\text{aborts while waiting for locks}] = \text{ex\_wait\_fail} + \text{sh\_wait\_fail}
\]

### 7.14.33 The ndbinfo logbuffers Table

The `logbuffer` table provides information on NDB Cluster log buffer usage.

The `logbuffers` table contains the following columns:

- **node\_id**
  - The ID of this data node.
- **log\_type**
  - Type of log. One of: \text{REDO, DD\_UNDO, BACKUP\_DATA}, or \text{BACKUP\_LOG}.
- **log\_id**
  - The log ID; for Disk Data undo log files, this is the same as the value shown in the \text{LOGFILE\_GROUP\_NUMBER} column of the \text{INFORMATION\_SCHEMA\_FILES} table as well as the value shown for the \text{log\_id} column of the ndbinfo logspaces table.
- **log\_part**
  - The log part number.
- **total**
  - Total space available for this log.
- **used**
  - Space used by this log.

**Notes**

`logbuffers` table rows reflecting two additional log types are available when performing an NDB backup. One of these rows has the log type \text{BACKUP\_DATA}, which shows the amount of data buffer used during backup to copy fragments to backup files. The other row has the log type \text{BACKUP\_LOG}.
LOG, which displays the amount of log buffer used during the backup to record changes made after the backup has started. One each of these log_type rows is shown in the logbuffers table for each data node in the cluster. These rows are not present unless an NDB backup is currently being performed.

### 7.14.34 The ndbinfo logspaces Table

This table provides information about NDB Cluster log space usage.

The logspaces table contains the following columns:

- **node_id**
  - The ID of this data node.

- **log_type**
  - Type of log; one of: REDO or DD-UNDO.

- **node_id**
  - The log ID; for Disk Data undo log files, this is the same as the value shown in the LOGFILE_GROUP_NUMBER column of the INFORMATION_SCHEMA.FILES table, as well as the value shown for the log_id column of the ndbinfo logbuffers table.

- **log_part**
  - The log part number.

- **total**
  - Total space available for this log.

- **used**
  - Space used by this log.

### 7.14.35 The ndbinfo membership Table

The membership table describes the view that each data node has of all the others in the cluster, including node group membership, president node, arbitrator, arbitrator successor, arbitrator connection states, and other information.

The membership table contains the following columns:

- **node_id**
  - This node's node ID

- **group_id**
  - Node group to which this node belongs

- **left_node**
  - Node ID of the previous node

- **right_node**
  - Node ID of the next node

- **president**
  -
The ndbinfo membership Table

President's node ID

• successor
  Node ID of successor to president

• succession_order
  Order in which this node succeeds to presidency

• Conf_HB_order

• arbitrator
  Node ID of arbitrator

• arb_ticket
  Internal identifier used to track arbitration

• arb_state
  Arbitration state

• arb_connected
  Whether this node is connected to the arbitrator; either of Yes or No

• connected_rank1_arbs
  Connected arbitrators of rank 1

• connected_rank2_arbs
  Connected arbitrators of rank 1

Notes

The node ID and node group ID are the same as reported by ndb_mgm -e "SHOW".

left_node and right_node are defined in terms of a model that connects all data nodes in a circle, in order of their node IDs, similar to the ordering of the numbers on a clock dial, as shown here:

Figure 7.1 Circular Arrangement of NDB Cluster Nodes

In this example, we have 8 data nodes, numbered 5, 6, 7, 8, 12, 13, 14, and 15, ordered clockwise in a circle. We determine “left” and “right” from the interior of the circle. The node to the left of node 5 is...
node 15, and the node to the right of node 5 is node 6. You can see all these relationships by running
the following query and observing the output:

```sql
mysql> SELECT node_id, left_node, right_node
-> FROM ndbinfo.membership;
```

```
+---------+-----------+------------+
| node_id | left_node | right_node |
+---------+-----------+------------+
|       5 |        15 |          6 |
|       6 |         5 |          7 |
|       7 |         6 |          8 |
|       8 |         7 |         12 |
|      12 |         8 |         13 |
|      13 |        12 |         14 |
|      14 |        13 |         15 |
|      15 |        14 |          5 |
+---------+-----------+------------+
8 rows in set (0.00 sec)
```

The designations “left” and “right” are used in the event log in the same way.

The `president` node is the node viewed by the current node as responsible for setting an arbitrator
(see NDB Cluster Start Phases). If the president fails or becomes disconnected, the current node
expects the node whose ID is shown in the `successor` column to become the new president. The
`succession_order` column shows the place in the succession queue that the current node views
itself as having.

In a normal NDB Cluster, all data nodes should see the same node as `president`, and the same
node (other than the president) as its `successor`. In addition, the current president should see itself as
1 in the order of succession, the `successor` node should see itself as 2, and so on.

All nodes should show the same `arb_ticket` values as well as the same `arb_state` values.
Possible `arb_state` values are `ARBIT_NULL`, `ARBIT_INIT`, `ARBIT_FIND`, `ARBIT_PREP1`,
`ARBIT_PREP2`, `ARBIT_START`, `ARBIT_RUN`, `ARBIT_CHOOSE`, `ARBIT_CRASH`, and `UNKNOWN`.

`arb_connected` shows whether this node is connected to the node shown as this node’s
`arbitor`.

The `connected_rank1_arbs` and `connected_rank2_arbs` columns each display a list of 0 or
more arbitrators having an `ArbitrationRank` equal to 1, or to 2, respectively.

**Note**
Both management nodes and API nodes are eligible to become arbitrators.

### 7.14.36 The ndbinfo memoryusage Table

Querying this table provides information similar to that provided by the `ALL REPORT MemoryUsage`
command in the `ndb_mgm` client, or logged by `ALL DUMP 1000`.

The `memoryusage` table contains the following columns:

- **node_id**
  The node ID of this data node.

- **memory_type**
  One of `Data memory`, `Index memory`, or `Long message buffer`.

- **used**
  Number of bytes currently used for data memory or index memory by this data node.
The ndbinfo memory_per_fragment Table

- **used_pages**
  Number of pages currently used for data memory or index memory by this data node; see text.

- **total**
  Total number of bytes of data memory or index memory available for this data node; see text.

- **total_pages**
  Total number of memory pages available for data memory or index memory on this data node; see text.

Notes

The **total** column represents the total amount of memory in bytes available for the given resource (data memory or index memory) on a particular data node. This number should be approximately equal to the setting of the corresponding configuration parameter in the `config.ini` file.

Suppose that the cluster has 2 data nodes having node IDs 5 and 6, and the `config.ini` file contains the following:

```
[ndbd default]
DataMemory = 1G
IndexMemory = 1G
```

Suppose also that the value of the `LongMessageBuffer` configuration parameter is allowed to assume its default (64 MB).

The following query shows approximately the same values:

```sql
mysql> SELECT node_id, memory_type, total
    > FROM ndbinfo.memoryusage;
+---------+---------------------+------------+
| node_id | memory_type         | total      |
+---------+---------------------+------------+
|       5 | Data memory         | 1073741824 |
|       5 | Index memory        | 1074003968 |
|       5 | Long message buffer |   67108864 |
|       6 | Data memory         | 1073741824 |
|       6 | Index memory        | 1074003968 |
|       6 | Long message buffer |   67108864 |
+---------+---------------------+------------+
6 rows in set (0.00 sec)
```

In this case, the **total** column values for index memory are slightly higher than the value set of `IndexMemory` due to internal rounding.

For the **used_pages** and **total_pages** columns, resources are measured in pages, which are 32K in size for `DataMemory` and 8K for `IndexMemory`. For long message buffer memory, the page size is 256 bytes.

7.14.37 The ndbinfo memory_per_fragment Table

The **memory_per_fragment** table provides information about the usage of memory by individual fragments.

The **memory_per_fragment** table contains the following columns:

- **fq_name**
  Name of this fragment

- **parent_fq_name**
The ndbinfo memory_per_fragment Table

Name of this fragment's parent

- **type**
  Type of object; see text for possible values

- **table_id**
  Table ID for this table

- **node_id**
  Node ID for this node

- **block_instance**
  Kernel block instance ID

- **fragment_num**
  Fragment ID (number)

- **fixed_elem_alloc_bytes**
  Number of bytes allocated for fixed-sized elements

- **fixed_elem_free_bytes**
  Free bytes remaining in pages allocated to fixed-size elements

- **fixed_elem_size_bytes**
  Length of each fixed-size element in bytes

- **fixed_elem_count**
  Number of fixed-size elements

- **fixed_elem_free_count**
  Number of free rows for fixed-size elements

- **var Elem Alloc Bytes**
  Number of bytes allocated for variable-size elements

- **var Elem Free Bytes**
  Free bytes remaining in pages allocated to variable-size elements

- **var Elem Count**
  Number of variable-size elements

- **hash_index_alloc_bytes**
  Number of bytes allocated to hash indexes

Notes

The **type** column from this table shows the dictionary object type used for this fragment (Object::Type, in the NDB API), and can take any one of the values shown in the following list:
The ndbinfo nodes Table

- System table
- User table
- Unique hash index
- Hash index
- Unique ordered index
- Ordered index
- Hash index trigger
- Subscription trigger
- Read only constraint
- Index trigger
- Reorganize trigger
- Tablespace
- Log file group
- Data file
- Undo file
- Hash map
- Foreign key definition
- Foreign key parent trigger
- Foreign key child trigger
- Schema transaction

You can also obtain this list by executing `SELECT * FROM ndbinfo.dict_obj_types` in the mysql client.

The `block_instance` column provides the NDB kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

### 7.14.38 The ndbinfo nodes Table

This table contains information on the status of data nodes. For each data node that is running in the cluster, a corresponding row in this table provides the node's node ID, status, and uptime. For nodes that are starting, it also shows the current start phase.

The `nodes` table contains the following columns:

- **node_id**
  The data node's unique node ID in the cluster.

- **uptime**
  Time since the node was last started, in seconds.

- **status**

Current status of the data node; see text for possible values.

- **start_phase**
  If the data node is starting, the current start phase.

- **config_generation**
  The version of the cluster configuration file in use on this data node.

**Notes**

The `uptime` column shows the time in seconds that this node has been running since it was last started or restarted. This is a BIGINT value. This figure includes the time actually needed to start the node; in other words, this counter starts running the moment that `ndbd` or `ndbmtd` is first invoked; thus, even for a node that has not yet finished starting, `uptime` may show a nonzero value.

The `status` column shows the node's current status. This is one of: NOTHING, CMVMI, STARTING, STARTED, SINGLEUSER, STOPPING_1, STOPPING_2, STOPPING_3, or STOPPING_4. When the status is STARTING, you can see the current start phase in the `start_phase` column (see later in this section). SINGLEUSER is displayed in the `status` column for all data nodes when the cluster is in single user mode (see Section 7.6, “NDB Cluster Single User Mode”). Seeing one of the STOPPING states does not necessarily mean that the node is shutting down but can mean rather that it is entering a new state. For example, if you put the cluster in single user mode, you can sometimes see data nodes report their state briefly as STOPPING_2 before the status changes to SINGLEUSER.

The `start_phase` column uses the same range of values as those used in the output of the `ndb_mgm client node_id STATUS` command (see Section 7.1, “Commands in the NDB Cluster Management Client”). If the node is not currently starting, then this column shows 0. For a listing of NDB Cluster start phases with descriptions, see Section 7.4, “Summary of NDB Cluster Start Phases”.

The `config_generation` column shows which version of the cluster configuration is in effect on each data node. This can be useful when performing a rolling restart of the cluster in order to make changes in configuration parameters. For example, from the output of the following `SELECT` statement, you can see that node 3 is not yet using the latest version of the cluster configuration (6) although nodes 1, 2, and 4 are doing so:

```sql
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
<table>
<thead>
<tr>
<th>node_id</th>
<th>uptime</th>
<th>status</th>
<th>start_phase</th>
<th>config_generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10462</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10460</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10457</td>
<td>STARTED</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>10455</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
+---------+--------+---------+-------------+-------------------+
2 rows in set (0.04 sec)
```

Therefore, for the case just shown, you should restart node 3 to complete the rolling restart of the cluster.

Nodes that are stopped are not accounted for in this table. Suppose that you have an NDB Cluster with 4 data nodes (node IDs 1, 2, 3 and 4), and all nodes are running normally, then this table contains 4 rows, 1 for each data node:

```sql
mysql> USE ndbinfo;
Database changed
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
<table>
<thead>
<tr>
<th>node_id</th>
<th>uptime</th>
<th>status</th>
<th>start_phase</th>
<th>config_generation</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>10462</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>2</td>
<td>10460</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
<tr>
<td>3</td>
<td>10457</td>
<td>STARTED</td>
<td>0</td>
<td>5</td>
</tr>
<tr>
<td>4</td>
<td>10455</td>
<td>STARTED</td>
<td>0</td>
<td>6</td>
</tr>
</tbody>
</table>
+---------+--------+---------+-------------+-------------------+
```
The ndbinfo operations_per_fragment Table

|       1 |  11776 | STARTED |           0 |                 6 |
|       2 |  11774 | STARTED |           0 |                 6 |
|       3 |  11771 | STARTED |           0 |                 6 |
|       4 |  11769 | STARTED |           0 |                 6 |

4 rows in set (0.04 sec)

If you shut down one of the nodes, only the nodes that are still running are represented in the output of this SELECT statement, as shown here:

```
ndb_mgm> 2 STOP
Node 2: Node shutdown initiated
Node 2: Node shutdown completed.
Node 2 has shutdown.
```

```
mysql> SELECT * FROM nodes;
+---------+--------+---------+-------------+-------------------+
| node_id | uptime | status  | start_phase | config_generation |
+---------+--------+---------+-------------+-------------------+
|       1 |  11807 | STARTED |           0 |                 6 |
|       3 |  11802 | STARTED |           0 |                 6 |
|       4 |  11800 | STARTED |           0 |                 6 |
+---------+--------+---------+-------------+-------------------+
3 rows in set (0.02 sec)
```

7.14.39 The ndbinfo operations_per_fragment Table

The operations_per_fragment table provides information about the operations performed on individual fragments and fragment replicas, as well as about some of the results from these operations.

The operations_per_fragment table contains the following columns:

- **fq_name**
  Name of this fragment

- **parent_fq_name**
  Name of this fragment's parent

- **type**
  Type of object; see text for possible values

- **table_id**
  Table ID for this table

- **node_id**
  Node ID for this node

- **block_instance**
  Kernel block instance ID

- **fragment_num**
  Fragment ID (number)

- **tot_key_reads**
  Total number of key reads for this fragment replica

- **tot_key_inserts**
  Total number of key inserts for this fragment replica
The ndbinfo operations_per_fragment Table

- **tot_key_updates**
  Total number of key updates for this fragment replica

- **tot_key_writes**
  Total number of key writes for this fragment replica

- **tot_key_deletes**
  Total number of key deletes for this fragment replica

- **tot_key_refs**
  Number of key operations refused

- **tot_key_attrinfo_bytes**
  Total size of all attrinfo attributes

- **tot_key_keyinfo_bytes**
  Total size of all keyinfo attributes

- **tot_key_prog_bytes**
  Total size of all interpreted programs carried by attrinfo attributes

- **tot_key_inst_exec**
  Total number of instructions executed by interpreted programs for key operations

- **tot_key_bytes_returned**
  Total size of all data and metadata returned from key read operations

- **tot_frag_scans**
  Total number of scans performed on this fragment replica

- **tot_scan_rows_examined**
  Total number of rows examined by scans

- **tot_scan_rows_returned**
  Total number of rows returned to client

- **tot_scan_bytes_returned**
  Total size of data and metadata returned to the client

- **tot_scan_prog_bytes**
  Total size of interpreted programs for scan operations

- **tot_scan_bound_bytes**
  Total size of all bounds used in ordered index scans

- **tot_scan_inst_exec**
  Total number of instructions executed by interpreted programs for scan operations
The ndbinfo operations_per_fragment Table

Total number of instructions executed for scans

- **tot_qd_frag_scans**
  Number of times that scans of this fragment replica have been queued

- **conc_frag_scans**
  Number of scans currently active on this fragment replica (excluding queued scans)

- **conc_qd_frag_scans**
  Number of scans currently queued for this fragment replica

- **tot_commits**
  Total number of row changes committed to this fragment replica

Notes

The **fq_name** contains the fully qualified name of the schema object to which this fragment replica belongs. This currently has the following formats:

- **Base table**: `DbName/def/TblName`
- **BLOB table**: `DbName/def/NDB$BLOB_BaseTblId_ColNo`
- **Ordered index**: `sys/def/BaseTblId/IndexName`
- **Unique index**: `sys/def/BaseTblId/IndexName$unique`

The `$unique` suffix shown for unique indexes is added by `mysqld`; for an index created by a different NDB API client application, this may differ, or not be present.

The syntax just shown for fully qualified object names is an internal interface which is subject to change in future releases.

Consider a table **t1** created and modified by the following SQL statements:

```sql
CREATE DATABASE mydb;
USE mydb;
CREATE TABLE t1 (  
  a INT NOT NULL,  
  b INT NOT NULL,  
  t TEXT NOT NULL,  
  PRIMARY KEY (b)  
) ENGINE=ndbcluster;
CREATE UNIQUE INDEX ix1 ON t1(b) USING HASH;
```

If **t1** is assigned table ID 11, this yields the **fq_name** values shown here:

- **Base table**: `mydb/def/t1`
- **BLOB table**: `mydb/def/NDB$BLOB_11_2`
- **Ordered index (primary key)**: `sys/def/11/PRIMARY`
- **Unique index**: `sys/def/11/ix1$unique`

For indexes or **BLOB** tables, the **parent_fq_name** column contains the **fq_name** of the corresponding base table. For base tables, this column is always **NULL**.

The **type** column shows the schema object type used for this fragment, which can take any one of the values **System table**, **User table**, **Unique hash index**, or **Ordered index**. **BLOB** tables are shown as **User table**.
The `ndbinfo pgman_time_track_stats` Table

The `table_id` column value is unique at any given time, but can be reused if the corresponding object has been deleted. The same ID can be seen using the `ndb_show_tables` utility.

The `block_instance` column shows which LDM instance this fragment replica belongs to. You can use this to obtain information about specific threads from the `threadblocks` table. The first such instance is always numbered 0.

Since there are typically two fragment replicas, and assuming that this is so, each `fragment_num` value should appear twice in the table, on two different data nodes from the same node group.

Since NDB does not use single-key access for ordered indexes, the counts for `tot_key_reads`, `tot_key_inserts`, `tot_key_updates`, `tot_key_writes`, and `tot_key_deletes` are not incremented by ordered index operations.

**Note**

When using `tot_key_writes`, you should keep in mind that a write operation in this context updates the row if the key exists, and inserts a new row otherwise. (One use of this is in the NDB implementation of the `REPLACE` SQL statement.)

The `tot_key_refs` column shows the number of key operations refused by the LDM. Generally, such a refusal is due to duplicate keys (inserts), Key not found errors (updates, deletes, and reads), or the operation was rejected by an interpreted program used as a predicate on the row matching the key.

The `attrinfo` and `keyinfo` attributes counted by the `tot_key_attrinfo_bytes` and `tot_key_keyinfo_bytes` columns are attributes of an LQHKEYREQ signal (see The NDB Communication Protocol) used to initiate a key operation by the LDM. An `attrinfo` typically contains tuple field values (inserts and updates) or projection specifications (for reads); `keyinfo` contains the primary or unique key needed to locate a given tuple in this schema object.

The value shown by `tot_frag_scans` includes both full scans (that examine every row) and scans of subsets. Unique indexes and BLOB tables are never scanned, so this value, like other scan-related counts, is 0 for fragment replicas of these.

`tot_scan_rows_examined` may display less than the total number of rows in a given fragment replica, since ordered index scans can be limited by bounds. In addition, a client may choose to end a scan before all potentially matching rows have been examined; this occurs when using an SQL statement containing a LIMIT or EXISTS clause, for example. `tot_scan_rows_returned` is always less than or equal to `tot_scan_rows_examined`.

`tot_scan_bytes_returned` includes, in the case of pushed joins, projections returned to the DBSPJ block in the NDB kernel.

`tot_qd_frag_scans` can be effected by the setting for the `MaxParallelScansPerFragment` data node configuration parameter, which limits the number of scans that may execute concurrently on a single fragment replica.

### 7.14.40 The `ndbinfo pgman_time_track_stats` Table

This table provides information regarding the latency of disk operations for NDB Cluster Disk Data tablespaces.

The `pgman_time_track_stats` table contains the following columns:

- **node_id**
  
  Unique node ID of this node in the cluster

- **block_number**
The ndbinfo processes Table

<table>
<thead>
<tr>
<th>Block number (from blocks table)</th>
</tr>
</thead>
<tbody>
<tr>
<td>• block_instance</td>
</tr>
<tr>
<td>Block instance number</td>
</tr>
<tr>
<td>• upper_bound</td>
</tr>
<tr>
<td>Upper bound</td>
</tr>
<tr>
<td>• page_reads</td>
</tr>
<tr>
<td>Page read latency (ms)</td>
</tr>
<tr>
<td>• page_writes</td>
</tr>
<tr>
<td>Page write latency (ms)</td>
</tr>
<tr>
<td>• log_waits</td>
</tr>
<tr>
<td>Log wait latency (ms)</td>
</tr>
<tr>
<td>• get_page</td>
</tr>
<tr>
<td>Latency of get_page() calls (ms)</td>
</tr>
</tbody>
</table>

Notes

The read latency (page_reads column) measures the time from when the read request is sent to the file system thread until the read is complete and has been reported back to the execution thread. The write latency (page_writes) is calculated in a similar fashion. The size of the page read to or written from a Disk Data tablespace is always 32 KB.

Log wait latency (log_waits column) is the length of time a page write must wait for the undo log to be flushed, which must be done prior to each page write.

The pgman_time_track_stats table was added in NDB 8.0.19.

7.14.41 The ndbinfo processes Table

This table contains information about NDB Cluster node processes; each node is represented by the row in the table. Only nodes that are connected to the cluster are shown in this table. You can obtain information about nodes that are configured but not connected to the cluster from the nodes and config_nodes tables.

The processes table contains the following columns:

• node_id
  The node’s unique node ID in the cluster

• node_type
  Type of node (management, data, or API node; see text)

• node_version
  Version of the NDB software program running on this node.

• process_id
This node's process ID

- `angel_process_id`
  
  Process ID of this node's angel process

- `process_name`
  
  Name of the executable

- `service_URI`
  
  Service URI of this node (see text)

**Notes**

`node_id` is the ID assigned to this node in the cluster.

The `node_type` column displays one of the following three values:

- **MGM**: Management node.
- **NDB**: Data node.
- **API**: API or SQL node.

For an executable shipped with the NDB Cluster distribution, `node_version` shows the software Cluster version string, such as `8.0.24-ndb-8.0.24`.

`process_id` is the node executable's process ID as shown by the host operating system using a process display application such as `top` on Linux, or the Task Manager on Windows platforms.

`angel_process_id` is the system process ID for the node's angel process, which ensures that a data node or SQL is automatically restarted in cases of failures. For management nodes and API nodes other than SQL nodes, the value of this column is `NULL`.

The `process_name` column shows the name of the running executable. For management nodes, this is `ndb_mgmd`. For data nodes, this is `ndbd` (single-threaded) or `ndbmtd` (multithreaded). For SQL nodes, this is `mysqld`. For other types of API nodes, it is the name of the executable program connected to the cluster; NDB API applications can set a custom value for this using `Ndb_cluster_connection::set_name()`.

`service_URI` shows the service network address. For management nodes and data nodes, the scheme used is `ndb://`. For management nodes and SQL nodes use `ndb://` for the scheme; NDB API applications can set this to a custom value using `Ndb_cluster_connection::set_service_uri()`. Regardless of the node type, the scheme is followed by the IP address used by the NDB transporter for the node in question. For management nodes and SQL nodes, this address includes the port number (usually 1186 for management nodes and 3306 for SQL nodes). If the SQL node was started with the `bind_address` system variable set, this address is used instead of the transporter address, unless the bind address is set to `*`, `0.0.0.0`, or `::`.

Additional path information may be included in the `service_URI` value for an SQL node reflecting various configuration options. For example, `mysql://198.51.100.3/tmp/mysql.sock` indicates that the SQL node was started with the `skip_networking` system variable enabled, and `mysql://198.51.100.3:3306/?server-id=1` shows that replication is enabled for this SQL node.

**7.14.42 The ndbinfo resources Table**
The ndbinfo resources Table

This table provides information about data node resource availability and usage. These resources are sometimes known as super-pools.

The resources table contains the following columns:

- **node_id**
  The unique node ID of this data node.

- **resource_name**
  Name of the resource; see text.

- **reserved**
  The amount reserved for this resource.

- **used**
  The amount actually used by this resource.

- **max**
  The maximum amount of this resource used, since the node was last started.

Notes

The resource_name can be any one of the names shown in the following table:

- **RESERVED**: Reserved by the system; cannot be overridden.

- **DISK_OPERATIONS**: If a log file group is allocated, the size of the undo log buffer is used to set the size of this resource. This resource is used only to allocate the undo log buffer for an undo log file group; there can only be one such group. Overallocation occurs as needed by CREATE LOGFILE GROUP.

- **DISK_RECORDS**: Records allocated for Disk Data operations.

- **DATA_MEMORY**: Used for main memory tuples, indexes, and hash indexes. Sum of DataMemory and IndexMemory, plus 8 pages of 32 KB each if IndexMemory has been set. Cannot be overallocated.

- **JOBBUFFER**: Used for allocating job buffers by the NDB scheduler; cannot be overallocated. This is approximately 2 MB per thread plus a 1 MB buffer in both directions for all threads that can communicate. For large configurations this consume several GB.

- **FILE_BUFFERS**: Used by the redo log handler in the DBLQH kernel block; cannot be overallocated. Size is \( \text{NoOfFragmentLogParts} \times \text{RedoBuffer} \), plus 1 MB per log file part.

- **TRANSPORTER_BUFFERS**: Used for send buffers by ndbmtd; the sum of TotalSendBufferMemory and ExtraSendBufferMemory. This resource that can be overallocated by up to 25 percent. TotalSendBufferMemory is calculated by summing the send buffer memory per node, the default value of which is 2 MB. Thus, in a system having four data nodes and eight API nodes, the data nodes have 12 * 2 MB send buffer memory. ExtraSendBufferMemory is used by ndbmtd and amounts to 2 MB extra memory per thread. Thus, with 4 LDM threads, 2 TC threads, 1 main thread, 1 replication thread, and 2 receive threads, ExtraSendBufferMemory is 10 * 2 MB. Overallocation of this resource can be performed by setting the SharedGlobalMemory data node configuration parameter.

- **DISK_PAGE_BUFFER**: Used for the disk page buffer; determined by the DiskPageBufferMemory configuration parameter. Cannot be overallocated.

- **QUERY_MEMORY**: Used by the DBSPJ kernel block.
• SCHEMA_TRANS_MEMORY: Minimum is 2 MB; can be overallocated to use any remaining available memory.

7.14.43 The ndbinfo restart_info Table

The restart_info table contains information about node restart operations. Each entry in the table corresponds to a node restart status report in real time from a data node with the given node ID. Only the most recent report for any given node is shown.

The restart_info table contains the following columns:

• node_id
  Node ID in the cluster

• node_restart_status
  Node status; see text for values. Each of these corresponds to a possible value of node_restart_status_int.

• node_restart_status_int
  Node status code; see text for values.

• secs_to_complete_node_failure
  Time in seconds to complete node failure handling

• secs_to_allocate_node_id
  Time in seconds from node failure completion to allocation of node ID

• secs_to_include_in_heartbeat_protocol
  Time in seconds from allocation of node ID to inclusion in heartbeat protocol

• secs_until_wait_for_ndbcntr_master
  Time in seconds from being included in heartbeat protocol until waiting for NDNCNTR master began

• secs_wait_for_ndbcntr_master
  Time in seconds spent waiting to be accepted by NDNCNTR master for starting

• secs_to_get_start_permitted
  Time in seconds elapsed from receiving of permission for start from master until all nodes have accepted start of this node

• secs_to_wait_for_lcp_for_copy_meta_data
  Time in seconds spent waiting for LCP completion before copying metadata

• secs_to_copy_meta_data
  Time in seconds required to copy metadata from master to newly starting node

• secs_to_include_node
  Time in seconds waited for GCP and inclusion of all nodes into protocols

• secs_starting_node_to_request_local_recovery
  Time in seconds that the node just starting spent waiting to request local recovery
The ndbinfo restart_info Table

- **secs_for_local_recovery**
  Time in seconds required for local recovery by node just starting

- **secs_restore_fragments**
  Time in seconds required to restore fragments from LCP files

- **secs_undo_disk_data**
  Time in seconds required to execute undo log on disk data part of records

- **secs_exec_redo_log**
  Time in seconds required to execute redo log on all restored fragments

- **secs_index_rebuild**
  Time in seconds required to rebuild indexes on restored fragments

- **secs_to_synchronize_starting_node**
  Time in seconds required to synchronize starting node from live nodes

- **secs_wait_lcp_for_restart**
  Time in seconds required for LCP start and completion before restart was completed

- **secs_wait_subscription_handover**
  Time in seconds spent waiting for handover of replication subscriptions

- **total_restart_secs**
  Total number of seconds from node failure until node is started again

Notes

The following list contains values defined for the **node_restart_status_int** column with their internal status names (in parentheses), and the corresponding messages shown in the **node_restart_status** column:

- **0 (ALLOCATED_NODE_ID)**
  Allocated node id

- **1 (INCLUDED_IN_HB_PROTOCOL)**
  Included in heartbeat protocol

- **2 (NDBCNTR_START_WAIT)**
  Wait for NDBCNTR master to permit us to start

- **3 (NDBCNTR_STARTED)**
  NDBCNTR master permitted us to start

- **4 (START_PERMITTED)**
  All nodes permitted us to start

- **5 (WAIT_LCP_TO_COPY_DICT)**
The ndbinfo restart_info Table

Wait for LCP completion to start copying metadata
- 6 (COPY_DICT_TO_STARTING_NODE)
  Copying metadata to starting node

Include node in LCP and GCP protocols
- 7 (INCLUDE_NODE_IN_LCP_AND_GCP)

Restore fragments ongoing
- 8 (LOCAL_RECOVERY_STARTED)

Synchronizing starting node with live nodes
- 9 (COPY_FRAGMENTS_STARTED)

Wait for LCP to ensure durability
- 10 (WAIT_LCP_FOR_RESTART)

Wait for handover of subscriptions
- 11 (WAIT_SUMA_HANDOVER)

Restart completed
- 12 (RESTART_COMPLETED)

Node failed, failure handling in progress
- 13 (NODE_FAILED)

Node failure handling completed
- 14 (NODE_FAILURE_COMPLETED)

All nodes permitted us to start
- 15 (NODE_GETTING_PERMIT)

Include node in LCP and GCP protocols
- 16 (NODE_GETTING_INCLUDED)

Synchronizing starting node with live nodes
- 17 (NODE_GETTING_SYNCHED)

Restart completed
- 18 (NODE_GETTING_LCP_WAITED)
  [none]

Node not restarted yet
- 19 (NODE_ACTIVE)

Node not restarted yet
- 20 (NOT_DEFINED_IN_CLUSTER)
  [none]

Node not restarted yet
- 21 (NODE_NOT_RESTARTED_YET)
Initial state

Status numbers 0 through 12 apply on master nodes only; the remainder of those shown in the table apply to all restarting data nodes. Status numbers 13 and 14 define node failure states; 20 and 21 occur when no information about the restart of a given node is available.

See also Section 7.4, “Summary of NDB Cluster Start Phases”.

7.14.44 The ndbinfo server_locks Table

The server_locks table is similar in structure to the cluster_locks table, and provides a subset of the information found in the latter table, but which is specific to the SQL node (MySQL server) where it resides. (The cluster_locks table provides information about all locks in the cluster.) More precisely, server_locks contains information about locks requested by threads belonging to the current mysqld instance, and serves as a companion table to server_operations. This may be useful for correlating locking patterns with specific MySQL user sessions, queries, or use cases.

The server_locks table contains the following columns:

- mysql_connection_id
  MySQL connection ID
- node_id
  ID of reporting node
- block_instance
  ID of reporting LDM instance
- tableid
  ID of table containing this row
- fragmentid
  ID of fragment containing locked row
- rowid
  ID of locked row
- transid
  Transaction ID
- mode
  Lock request mode
- state
  Lock state
- detail
  Whether this is first holding lock in row lock queue
- op
  Operation type
The ndbinfo server_operations Table

- **duration_millis**
  Milliseconds spent waiting or holding lock
- **lock_num**
  ID of lock object
- **waiting_for**
  Waiting for lock with this ID

Notes

The `mysql_connection_id` column shows the MySQL connection or thread ID as shown by `SHOW PROCESSLIST`.

`block_instance` refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The `tableid` is assigned to the table by NDB; the same ID is used for this table in other ndbinfo tables, as well as in the output of `ndb_show_tables`.

The transaction ID shown in the `transid` column is the identifier generated by the NDB API for the transaction requesting or holding the current lock.

The `mode` column shows the lock mode, which is always one of `S` (shared lock) or `X` (exclusive lock). If a transaction has an exclusive lock on a given row, all other locks on that row have the same transaction ID.

The `state` column shows the lock state. Its value is always one of `H` (holding) or `W` (waiting). A waiting lock request waits for a lock held by a different transaction.

The `detail` column indicates whether this lock is the first holding lock in the affected row's lock queue, in which case it contains a `*` (asterisk character); otherwise, this column is empty. This information can be used to help identify the unique entries in a list of lock requests.

The `op` column shows the type of operation requesting the lock. This is always one of the values `READ`, `INSERT`, `UPDATE`, `DELETE`, `SCAN`, or `REFRESH`.

The `duration_millis` column shows the number of milliseconds for which this lock request has been waiting or holding the lock. This is reset to 0 when a lock is granted for a waiting request.

The lock ID (`lockid` column) is unique to this node and block instance.

If the `lock_state` column's value is `W`, this lock is waiting to be granted, and the `waiting_for` column shows the lock ID of the lock object this request is waiting for. Otherwise, `waiting_for` is empty. `waiting_for` can refer only to locks on the same row (as identified by `node_id`, `block_instance`, `tableid`, `fragmentid`, and `rowid`).

7.14.45 The ndbinfo server_operations Table

The `server_operations` table contains entries for all ongoing NDB operations that the current SQL node (MySQL Server) is currently involved in. It effectively is a subset of the `cluster_operations` table, in which operations for other SQL and API nodes are not shown.

The `server_operations` table contains the following columns:

- `mysql_connection_id`
The ndbinfo server_operations Table

MySQL Server connection ID

• node_id
  Node ID

• block_instance
  Block instance

• transid
  Transaction ID

• operation_type
  Operation type (see text for possible values)

• state
  Operation state (see text for possible values)

• tableid
  Table ID

• fragmentid
  Fragment ID

• client_node_id
  Client node ID

• client_block_ref
  Client block reference

• tc_node_id
  Transaction coordinator node ID

• tc_block_no
  Transaction coordinator block number

• tc_block_instance
  Transaction coordinator block instance

Notes

The mysql_connection_id is the same as the connection or session ID shown in the output of SHOW PROCESSLIST. It is obtained from the INFORMATION_SCHEMA table NDB_TRANSID_MYSQL_CONNECTION_MAP.

block_instance refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the threadblocks table.

The transaction ID (transid) is a unique 64-bit number which can be obtained using the NDB API's getTransactionId() method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)
The *operation_type* column can take any one of the values READ, READ-SH, READ-EX, INSERT, UPDATE, DELETE, WRITE, UNLOCK, REFRESH, SCAN, SCAN-SH, SCAN-EX, or <unknown>.

The *state* column can have any one of the values ABORT_QUEUE, ABORT_STOPPED, COMMITTED, COMMIT_QUEUE, COMMIT_STOPPED, COPY_CLOSE_STOPPED, COPY_FIRST_STOPPED, COPY_STOPPED, COPY_TUPKEY, IDLE, LOG_ABORT_QUEUE, LOG_COMMIT_QUEUE, LOG_COMMIT_QUERED_WAIT_SIGNAL, LOG_COMMIT_WRITTEN, LOG_COMMIT_WRITTEN_WAIT_SIGNAL, LOG_QUEUE, PREPARED, PREPARED_RECEIVED_COMMIT, SCAN_CHECK_STOPPED, SCAN_CLOSE_STOPPED, SCAN_FIRST_STOPPED, SCAN_RELEASE_STOPPED, SCAN_STATE_USED, SCAN_STOPPED, SCAN_TUPKEY, STOPPED, TC_NOT_CONNECTED, WAIT_ACC, WAIT_ACC_ABORT, WAIT_AI_AFTER_ABORT, WAIT_ATTR, WAIT_SCAN_AI, WAIT_TUP, WAIT_TUPKEYINFO, WAIT_TUP_COMMIT, or WAIT_TUP_TO_ABORT. (If the MySQL Server is running with *ndbinfo_show_hidden* enabled, you can view this list of states by selecting from the *ndb$dblqh_tcconnect_state* table, which is normally hidden.)

You can obtain the name of an *NDB* table from its table ID by checking the output of *ndb_show_tables*.

The *fragid* is the same as the partition number seen in the output of *ndb_desc --extra-partition-info* (short form *-p*).

In *client_node_id* and *client_block_ref*, *client* refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The *block_instance* and *tc_block_instance* column provide NDB kernel block instance numbers. You can use these to obtain information about specific threads from the *threadblocks* table.

### 7.14.46 The *ndbinfo server_transactions* Table

The *server_transactions* table is subset of the *cluster_transactions* table, but includes only those transactions in which the current SQL node (MySQL Server) is a participant, while including the relevant connection IDs.

The *server_transactions* table contains the following columns:

- **mysql_connection_id**
  
  MySQL Server connection ID

- **node_id**
  
  Transaction coordinator node ID

- **block_instance**
  
  Transaction coordinator block instance

- **transid**
  
  Transaction ID

- **state**
  
  Operation state (see text for possible values)

- **count_operations**
  
  Number of stateful operations in the transaction

- **outstanding_operations**
  
  Number of stateful operations in the transaction
The ndbinfo table_distribution_status Table

Operations still being executed by local data management layer (LQH blocks)

- **inactive_seconds**
  Time spent waiting for API

- **client_node_id**
  Client node ID

- **client_block_ref**
  Client block reference

**Notes**

The *mysql_connection_id* is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`. It is obtained from the `INFORMATION_SCHEMA` table `NDB_TRANSID_MYSQL_CONNECTION_MAP`.

*block_instance* refers to an instance of a kernel block. Together with the block name, this number can be used to look up a given instance in the `threadblocks` table.

The transaction ID (*transid*) is a unique 64-bit number which can be obtained using the NDB API's `getTransactionId()` method. (Currently, the MySQL Server does not expose the NDB API transaction ID of an ongoing transaction.)

The *state* column can have any one of the values `CS_ABORTING`, `CS_COMMITTING`, `CS_COMMIT_SENT`, `CS_COMPLETE_SENT`, `CS_COMMITTING`, `CS_CONNECTED`, `CS_DISCONNECTED`, `CS_FAIL_ABORTED`, `CS_FAIL_ABORTING`, `CS_FAIL_ABORTING`, `CS_FAIL_COMMITTED`, `CS_FAIL_COMMITTING`, `CS_FAIL_COMPLETED`, `CS_FAIL_PREPARED`, `CS_PREPARE_TO_COMMIT`, `CS_RECEIVING`, `CS_REC_COMMITTING`, `CS_RESTART`, `CS_SEND_FIRE_TRIG_REQ`, `CS_STARTED`, `CS_START_COMMITTING`, `CS_START_SCAN`, `CS_WAIT_ABORT_CONF`, `CS_WAIT_COMMIT_CONF`, `CS_WAIT_COMPLETE_CONF`, `CS_WAIT_FIRE_TRIG_REQ`. (If the MySQL Server is running with `ndbinfo_show_hidden` enabled, you can view this list of states by selecting from the `ndb $dbtc_apiconnect_state` table, which is normally hidden.)

In *client_node_id* and *client_block_ref*, *client* refers to an NDB Cluster API or SQL node (that is, an NDB API client or a MySQL Server attached to the cluster).

The *block_instance* column provides the DBTC kernel block instance number. You can use this to obtain information about specific threads from the `threadblocks` table.

### 7.14.47 The ndbinfo table_distribution_status Table

The `table_distribution_status` table provides information about the progress of table distribution for NDB tables.

The `table_distribution_status` table contains the following columns:

- **node_id**
  Node id

- **table_id**
  Table ID

- **tab_copy_status**
The ndbinfo table_fragments Table

Status of copying of table distribution data to disk; one of IDLE, SR_PHASE1_READ_PAGES, SR_PHASE2_READ_TABLE, SR_PHASE3_COPY_TABLE, REMOVE_NODE, LCP_READ_TABLE, COPY_TAB_REQ, COPY_NODE_STATE, ADD_TABLE_COORDINATOR (prior to NDB 8.0.23: ADD_TABLE_MASTER), ADD_TABLE_PARTICIPANT (prior to NDB 8.0.23: ADD_TABLE_SLAVE), INVALIDATE_NODE_LCP, ALTER_TABLE, COPY_TO_SAVE, or GET_TABINFO

• tab_update_status

Status of updating of table distribution data; one of IDLE, LOCAL_CHECKPOINT, LOCAL_CHECKPOINT_QUEUED, REMOVE_NODE, COPY_TAB_REQ, ADD_TABLE_MASTER, ADD_TABLE_SLAVE, INVALIDATE_NODE_LCP, or CALLBACK

• tab_lcp_status

Status of table LCP; one of ACTIVE (waiting for local checkpoint to be performed), WRITING_TO_FILE (checkpoint performed but not yet written to disk), or COMPLETED (checkpoint performed and persisted to disk)

• tab_status

Table internal status; one of ACTIVE (table exists), CREATING (table is being created), or DROPING (table is being dropped)

• tab_storage

Table recoverability; one of NORMAL (fully recoverable with redo logging and checkpointing), NOLOGGING (recoverable from node crash, empty following cluster crash), or TEMPORARY (not recoverable)

• tab_partitions

Number of partitions in table

• tab_fragments

Number of fragments in table; normally same as tab_partitions; for fully replicated tables equal to tab_partitions * [number of node groups]

• current_scan_count

Current number of active scans

• scan_count_wait

Current number of scans waiting to be performed before ALTER TABLE can complete.

• is_reorg_ongoing

Whether the table is currently being reorganized (1 if true)

7.14.48 The ndbinfo table_fragments Table

The table_fragments table provides information about the fragmentation, partitioning, distribution, and (internal) replication of NDB tables.

The table_fragments table contains the following columns:

• node_id

Node ID (DIH master)

• table_id
The ndbinfo table_info Table

The `table_info` table provides information about logging, checkpointing, distribution, and storage options in effect for individual NDB tables.

The `table_info` table contains the following columns:

- **table_id**
The ndbinfo table_replicas Table

Table ID

- **logged_table**
  Whether table is logged (1) or not (0)

- **row_contains_gci**
  Whether table rows contain GCI (1 true, 0 false)

- **row_contains_checksum**
  Whether table rows contain checksum (1 true, 0 false)

- **read_backup**
  If backup fragment replicas are read this is 1, otherwise 0

- **fully_replicated**
  If table is fully replicated this is 1, otherwise 0

- **storage_type**
  Table storage type; one of MEMORY or DISK

- **hasmap_id**
  Hashmap ID

- **partition_balance**
  Partition balance (fragment count type) used for table; one of FOR_RP_BY_NODE, FOR_RA_BY_NODE, FOR_RP_BY_LDM, or FOR_RA_BY_LDM

- **create_gci**
  GCI in which table was created

7.14.50 The ndbinfo table_replicas Table

The **table_replicas** table provides information about the copying, distribution, and checkpointing of NDB table fragments and fragment replicas.

The **table_replicas** table contains the following columns:

- **node_id**
  ID of the node from which data is fetched (DIH master)

- **table_id**
  Table ID

- **fragment_id**
  Fragment ID

- **initial_gci**
  Initial GCI for table

- **replica_node_id**
The ndbinfo tc_time_track_stats Table

- **ID of node where fragment replica is stored**
- **is_lcp_ongoing**
  Is 1 if LCP is ongoing on this fragment, 0 otherwise
- **num_crashed_replicas**
  Number of crashed fragment replica instances
- **last_max_gci_started**
  Highest GCI started in most recent LCP
- **last_max_gci_completed**
  Highest GCI completed in most recent LCP
- **last_lcp_id**
  ID of most recent LCP
- **prev_lcp_id**
  ID of previous LCP
- **prev_max_gci_started**
  Highest GCI started in previous LCP
- **prev_max_gci_completed**
  Highest GCI completed in previous LCP
- **last_create_gci**
  Last Create GCI of last crashed fragment replica instance
- **last_replica_gci**
  Last GCI of last crashed fragment replica instance
- **is_replica_alive**
  1 if this fragment replica is alive, 0 otherwise

### 7.14.51 The ndbinfo tc_time_track_stats Table

The `tc_time_track_stats` table provides time-tracking information obtained from the DBTC block (TC) instances in the data nodes, through API nodes access NDB. Each TC instance tracks latencies for a set of activities it undertakes on behalf of API nodes or other data nodes; these activities include transactions, transaction errors, key reads, key writes, unique index operations, failed key operations of any type, scans, failed scans, fragment scans, and failed fragment scans.

A set of counters is maintained for each activity, each counter covering a range of latencies less than or equal to an upper bound. At the conclusion of each activity, its latency is determined and the appropriate counter incremented. `tc_time_track_stats` presents this information as rows, with a row for each instance of the following:

- Data node, using its ID
- TC block instance
The ndbinfo tc_time_track_stats Table

- Other communicating data node or API node, using its ID
- Upper bound value

Each row contains a value for each activity type. This is the number of times that this activity occurred with a latency within the range specified by the row (that is, where the latency does not exceed the upper bound).

The `tc_time_track_stats` table contains the following columns:

- **node_id**
  Requesting node ID
- **block_number**
  TC block number
- **block_instance**
  TC block instance number
- **comm_node_id**
  Node ID of communicating API or data node
- **upper_bound**
  Upper bound of interval (in microseconds)
- **scans**
  Based on duration of successful scans from opening to closing, tracked against the API or data nodes requesting them.
- **scan_errors**
  Based on duration of failed scans from opening to closing, tracked against the API or data nodes requesting them.
- **scan_fragments**
  Based on duration of successful fragment scans from opening to closing, tracked against the data nodes executing them
- **scan_fragment_errors**
  Based on duration of failed fragment scans from opening to closing, tracked against the data nodes executing them
- **transactions**
  Based on duration of successful transactions from beginning until sending of commit ACK, tracked against the API or data nodes requesting them. Stateless transactions are not included.
- **transaction_errors**
  Based on duration of failing transactions from start to point of failure, tracked against the API or data nodes requesting them.
- **read_key_ops**
  Based on duration of successful primary key reads with locks. Tracked against both the API or data node requesting them and the data node executing them.
The ndbinfo threadblocks Table

- **write_key_ops**
  
  Based on duration of successful primary key writes, tracked against both the API or data node requesting them and the data node executing them.

- **index_key_ops**
  
  Based on duration of successful unique index key operations, tracked against both the API or data node requesting them and the data node executing reads of base tables.

- **key_op_errors**
  
  Based on duration of all unsuccessful key read or write operations, tracked against both the API or data node requesting them and the data node executing them.

Notes

The `block_instance` column provides the DBTC kernel block instance number. You can use this together with the block name to obtain information about specific threads from the `threadblocks` table.

7.14.52 The ndbinfo threadblocks Table

The `threadblocks` table associates data nodes, threads, and instances of NDB kernel blocks.

The `threadblocks` table contains the following columns:

- **node_id**
  
  Node ID

- **thr_no**
  
  Thread ID

- **block_name**
  
  Block name

- **block_instance**
  
  Block instance number

Notes

The value of the `block_name` in this table is one of the values found in the `block_name` column when selecting from the `ndbinfo.blocks` table. Although the list of possible values is static for a given NDB Cluster release, the list may vary between releases.

The `block_instance` column provides the kernel block instance number.

7.14.53 The ndbinfo threads Table

The `threads` table provides information about threads running in the NDB kernel.

The `threads` table contains the following columns:

- **node_id**
  
  ID of the node where the thread is running

- **thr_no**
The ndbinfo threadstat Table

- **Thread ID** (specific to this node)
  - **thread_name**
    - Thread name (type of thread)
  - **thread_description**
    - Thread (type) description

**Notes**

Sample output from a 2-node example cluster, including thread descriptions, is shown here:

```sql
mysql> SELECT * FROM threads;
```

<table>
<thead>
<tr>
<th>node_id</th>
<th>thr_no</th>
<th>thread_name</th>
<th>thread_description</th>
</tr>
</thead>
<tbody>
<tr>
<td>5</td>
<td>0</td>
<td>main</td>
<td>main thread, schema and distribution handling</td>
</tr>
<tr>
<td>5</td>
<td>1</td>
<td>rep</td>
<td>rep thread, asynch replication and proxy block handling</td>
</tr>
<tr>
<td>5</td>
<td>2</td>
<td>ldm</td>
<td>ldm thread, handling a set of data partitions</td>
</tr>
<tr>
<td>5</td>
<td>3</td>
<td>recv</td>
<td>receive thread, performing receive and polling for new receives</td>
</tr>
<tr>
<td>6</td>
<td>0</td>
<td>main</td>
<td>main thread, schema and distribution handling</td>
</tr>
<tr>
<td>6</td>
<td>1</td>
<td>rep</td>
<td>rep thread, asynch replication and proxy block handling</td>
</tr>
<tr>
<td>6</td>
<td>2</td>
<td>ldm</td>
<td>ldm thread, handling a set of data partitions</td>
</tr>
<tr>
<td>6</td>
<td>3</td>
<td>recv</td>
<td>receive thread, performing receive and polling for new receives</td>
</tr>
</tbody>
</table>

8 rows in set (0.01 sec)

NDB 8.0.23 introduces the possibility to set either of the `ThreadConfig` arguments `main` or `rep` to 0 while keeping the other at 1, in which case the thread name is `main_rep` and its description is main and rep thread, schema, distribution, proxy block and asynch replication handling. It is also possible beginning with NDB 8.0.23 to set both `main` and `rep` to 0, in which case the name of the resulting thread is shown in this table as `main_rep_recv`, and its description is main, rep and recv thread, schema, distribution, proxy block and asynch replication handling and handling receive and polling for new receives.

7.14.54 The ndbinfo threadstat Table

The `threadstat` table provides a rough snapshot of statistics for threads running in the NDB kernel.

The `threadstat` table contains the following columns:

- **node_id**
  - Node ID
- **thr_no**
  - Thread ID
- **thr_nm**
  - Thread name
- **c_loop**
  - Number of loops in main loop
- **c_exec**
  - Number of signals executed
- **c_wait**
  - Number of signals executed
The ndbinfo transporters Table

Number of times waiting for additional input

- c_l_sent_prioa
  Number of priority A signals sent to own node

- c_l_sent_priob
  Number of priority B signals sent to own node

- c_r_sent_prioa
  Number of priority A signals sent to remote node

- c_r_sent_priob
  Number of priority B signals sent to remote node

- os_tid
  OS thread ID

- os_now
  OS time (ms)

- os_ru_utime
  OS user CPU time (µs)

- os_ru_stime
  OS system CPU time (µs)

- os_ru_minflt
  OS page reclaims (soft page faults)

- os_ru_majflt
  OS page faults (hard page faults)

- os_ru_nvcsw
  OS voluntary context switches

- os_ru_nivcsw
  OS involuntary context switches

Notes

**os_time** uses the system `gettimeofday()` call.

The values of the **os_ru_utime, os_ru_stime, os_ru_minflt, os_ru_majflt, os_ru_nvcsw,** and **os_ru_nivcsw** columns are obtained using the system `getrusage()` call, or the equivalent.

Since this table contains counts taken at a given point in time, for best results it is necessary to query this table periodically and store the results in an intermediate table or tables. The MySQL Server’s Event Scheduler can be employed to automate such monitoring. For more information, see Using the Event Scheduler.

7.14.55 The ndbinfo transporters Table
This table contains information about NDB transporters.

The `transporters` table contains the following columns:

- `node_id`
  This data node's unique node ID in the cluster
- `remote_node_id`
  The remote data node's node ID
- `status`
  Status of the connection
- `remote_address`
  Name or IP address of the remote host
- `bytes_sent`
  Number of bytes sent using this connection
- `bytes_received`
  Number of bytes received using this connection
- `connect_count`
  Number of times connection established on this transporter
- `overloaded`
  1 if this transporter is currently overloaded, otherwise 0
- `overload_count`
  Number of times this transporter has entered overload state since connecting
- `slowdown`
  1 if this transporter is in slowdown state, otherwise 0
- `slowdown_count`
  Number of times this transporter has entered slowdown state since connecting

**Notes**

For each running data node in the cluster, the `transporters` table displays a row showing the status of each of that node's connections with all nodes in the cluster, including itself. This information is shown in the table's `status` column, which can have any one of the following values: `CONNECTING`, `CONNECTED`, `DISCONNECTING`, or `DISCONNECTED`.

Connections to API and management nodes which are configured but not currently connected to the cluster are shown with status `DISCONNECTED`. Rows where the `node_id` is that of a data node which is not currently connected are not shown in this table. (This is similar omission of disconnected nodes in the `ndbinfo.nodes` table.

The `remote_address` is the host name or address for the node whose ID is shown in the `remote_node_id` column. The `bytes_sent` from this node and `bytes_received` by this node are the numbers, respectively, of bytes sent and received by the node using this connection since it
was established. For nodes whose status is **CONNECTING** or **DISCONNECTED**, these columns always display **0**.

Assume you have a 5-node cluster consisting of 2 data nodes, 2 SQL nodes, and 1 management node, as shown in the output of the `SHOW` command in the `ndb_mgm` client:

```
ndb_mgm> SHOW
Connected to Management Server at: localhost:1186
Cluster Configuration
-----------
[ndbd(NDB)] 2 node(s)
id=1 @10.100.10.1 (8.0.24-ndb-8.0.24, Nodegroup: 0, *)
id=2 @10.100.10.2 (8.0.24-ndb-8.0.24, Nodegroup: 0)
[ndb_mgmd(MGM)] 1 node(s)
id=10 @10.100.10.10 (8.0.24-ndb-8.0.24)
[mysqld(API)] 2 node(s)
id=20 @10.100.10.20 (8.0.24-ndb-8.0.24)
id=21 @10.100.10.21 (8.0.24-ndb-8.0.24)
```

There are 10 rows in the `transporters` table—5 for the first data node, and 5 for the second—assuming that all data nodes are running, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
     > FROM ndbinfo.transporters;
+---------+----------------+---------------+
| node_id | remote_node_id | status        |
+---------+----------------+---------------+
|       1 |              1 | DISCONNECTED  |
|       1 |              2 | CONNECTED     |
|       1 |             10 | CONNECTED     |
|       1 |             20 | CONNECTED     |
|       1 |             21 | CONNECTED     |
|       2 |              1 | CONNECTED     |
|       2 |              2 | DISCONNECTED  |
|       2 |             10 | CONNECTED     |
|       2 |             20 | CONNECTED     |
|       2 |             21 | CONNECTED     |
+---------+----------------+---------------+
```

If you shut down one of the data nodes in this cluster using the command `2 STOP` in the `ndb_mgm` client, then repeat the previous query (again using the `mysql` client), this table now shows only 5 rows—one row for each connection from the remaining management node to another node, including both itself and the data node that is currently offline—and displays **CONNECTING** for the status of each remaining connection to the data node that is currently offline, as shown here:

```
mysql> SELECT node_id, remote_node_id, status
     > FROM ndbinfo.transporters;
+---------+----------------+---------------+
| node_id | remote_node_id | status        |
+---------+----------------+---------------+
|       1 |              1 | DISCONNECTED  |
|       1 |              2 | CONNECTING    |
|       1 |             10 | CONNECTED     |
|       1 |             20 | CONNECTED     |
|       1 |             21 | CONNECTED     |
+---------+----------------+---------------+
```

The `connect_count`, `overloaded`, `overload_count`, `slowdown`, and `slowdown_count` counters are reset on connection, and retain their values after the remote node disconnects. The `bytes_sent` and `bytes_received` counters are also reset on connection, and so retain their values following disconnection (until the next connection resets them).

The **overload** state referred to by the `overloaded` and `overload_count` columns occurs when this transporter’s send buffer contains more than `OverloadLimit` bytes (default is 80% of `SendBufferMemory`, that is, `0.8 * 2097152 = 1677721` bytes). When a given transporter is in a state
of overload, any new transaction that tries to use this transporter fails with Error 1218 (Send Buffers overloaded in NDB kernel). This affects both scans and primary key operations.

The slowdown state referenced by the slowdown and slowdown_count columns of this table occurs when the transporter's send buffer contains more than 60% of the overload limit (equal to 0.6 * 2097152 = 1258291 bytes by default). In this state, any new scan using this transporter has its batch size reduced to minimize the load on the transporter.

Common causes of send buffer slowdown or overloading include the following:

- Data size, in particular the quantity of data stored in TEXT columns or BLOB columns (or both types of columns)
- Having a data node (ndbd or ndbmtd) on the same host as an SQL node that is engaged in binary logging
- Large number of rows per transaction or transaction batch
- Configuration issues such as insufficient SendBufferMemory
- Hardware issues such as insufficient RAM or poor network connectivity

See also Section 5.3.14, “Configuring NDB Cluster Send Buffer Parameters”.

### 7.15 INFORMATION_SCHEMA Tables for NDB Cluster

Two INFORMATION_SCHEMA tables provide information that is of particular use when managing an NDB Cluster. The FILES table provides information about NDB Cluster Disk Data files (see Section 7.10.1, “NDB Cluster Disk Data Objects”). The ndb_transid_mysql_connection_map table provides a mapping between transactions, transaction coordinators, and API nodes.

Additional statistical and other data about NDB Cluster transactions, operations, threads, blocks, and other aspects of performance can be obtained from the tables in the ndbinfo database. For information about these tables, see Section 7.14, “ndbinfo: The NDB Cluster Information Database”.

### 7.16 Quick Reference: NDB Cluster SQL Statements

This section discusses several SQL statements that can prove useful in managing and monitoring a MySQL server that is connected to an NDB Cluster, and in some cases provide information about the cluster itself.

- **SHOW ENGINE NDB STATUS, SHOW ENGINE NDBCLUSTER STATUS**

The output of this statement contains information about the server's connection to the cluster, creation and usage of NDB Cluster objects, and binary logging for NDB Cluster replication.

See SHOW ENGINE Statement, for a usage example and more detailed information.

- **SHOW ENGINES**

This statement can be used to determine whether or not clustering support is enabled in the MySQL server, and if so, whether it is active.

See SHOW ENGINES Statement, for more detailed information.

**Note**

This statement does not support a LIKE clause. However, you can use LIKE to filter queries against the INFORMATION_SCHEMA.ENGINES table, as discussed in the next item.
• **SELECT * FROM INFORMATION_SCHEMA.ENGINES [WHERE ENGINE LIKE 'NDB%']**

This is the equivalent of **SHOW ENGINES**, but uses the **ENGINES** table of the **INFORMATION_SCHEMA** database. Unlike the case with the **SHOW ENGINES** statement, it is possible to filter the results using a **LIKE** clause, and to select specific columns to obtain information that may be of use in scripts. For example, the following query shows whether the server was built with **NDB** support and, if so, whether it is enabled:

```sql
mysql> SELECT ENGINE, SUPPORT FROM INFORMATION_SCHEMA.ENGINES
    -> WHERE ENGINE LIKE 'NDB%';
```

<table>
<thead>
<tr>
<th>ENGINE</th>
<th>SUPPORT</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndbcluster</td>
<td>YES</td>
</tr>
<tr>
<td>ndbinfo</td>
<td>YES</td>
</tr>
</tbody>
</table>

If **NDB** support is not enabled, the preceding query returns an empty set. See [The INFORMATION_SCHEMA ENGINES Table](#), for more information.

• **SHOW VARIABLES LIKE 'NDB%'**

This statement provides a list of most server system variables relating to the **NDB** storage engine, and their values, as shown here:

```sql
mysql> SHOW VARIABLES LIKE 'NDB%';
+--------------------------------------+--------------------------+
| Variable_name                        | Value                    |
+--------------------------------------+--------------------------+
| ndb_allow_copying_alter_table        | ON                       |
| ndb_autoincrement_prefetch_sz        | 512                      |
| ndb_batch_size                       | 32768                    |
| ndb_blob_read_batch_bytes            | 65536                    |
| ndb_blob_write_batch_bytes           | 65536                    |
| ndb_clear_apply_status               | ON                       |
| ndb_cluster_connection_pool          | 1                        |
| ndb_cluster_connection_pool_nodeids  |                          |
| ndb_connectstring                    | 127.0.0.1                |
| ndb_data_node_neighbour              | 0                        |
| ndb_default_column_format            | FIXED                    |
| ndb_deferred_constraints             | 0                        |
| ndb_distribution                     | KEYHASH                  |
| ndb_eventbuffer_free_percent         | 20                       |
| ndb_eventbuffer_max_alloc            | 0                        |
| ndb_extra_logging                    | 1                        |
| ndb_force_send                       | ON                       |
| ndb_fully_replicated                 | OFF                      |
| ndb_index_stat_option                | loop_enable=1000ms,loop_idle=1000ms,loop_busy=100ms,update_batch=1,read_batch=4,idle_batch=32,check_batch=8,check_delay=10m,delete_batch=8,clean_delay=1m,error_batch=4,error_delay=1m,evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 |
| ndb_join_pushdown                    | ON                       |
| ndb_log_apply_status                 | OFF                      |
| ndb_log_bin                          | OFF                      |
| ndb_log_binlog_index                 | ON                       |
| ndb_log_empty_epochs                 | OFF                      |
| ndb_log_empty_update                 | OFF                      |
| ndb_log_exclusive_reads              | OFF                      |
| ndb_log_orig                         | OFF                      |
| ndb_log_transaction_id               | OFF                      |
| ndb_log_update_as_write              | ON                       |
| ndb_log_update_minimal               | OFF                      |
| ndb_log_updated_only                 | ON                       |
| ndb_metadata_check                   | ON                       |
| ndb_metadata_check_interval          | 60                       |
| ndb_metadata_sync                    | OFF                      |
| ndb_mgmd_host                        | 127.0.0.1                |
| ndb_nodeid                           | 0                        |
| ndb_optimization_delay               | 10                       |
```
Quick Reference: NDB Cluster SQL Statements

<table>
<thead>
<tr>
<th>VARIABLE_NAME</th>
<th>VARIABLE_VALUE</th>
</tr>
</thead>
<tbody>
<tr>
<td>ndb_optimized_node_selection</td>
<td>3</td>
</tr>
<tr>
<td>ndb_read_backup</td>
<td>ON</td>
</tr>
<tr>
<td>ndb_recv_thread_activation_threshold</td>
<td>8</td>
</tr>
<tr>
<td>ndb_recv_thread_cpu_mask</td>
<td></td>
</tr>
<tr>
<td>ndb_report_thresh_binlog_epoch_slip</td>
<td>10</td>
</tr>
<tr>
<td>ndb_report_thresh_binlog_mem_usage</td>
<td>10</td>
</tr>
<tr>
<td>ndb_row_checksum</td>
<td>1</td>
</tr>
<tr>
<td>ndb_schema_dist_lock_wait_timeout</td>
<td>30</td>
</tr>
<tr>
<td>ndb_schema_dist_timeout</td>
<td>120</td>
</tr>
<tr>
<td>ndb_schema_dist_upgrade_allowed</td>
<td>ON</td>
</tr>
<tr>
<td>ndb_show_foreign_key_mock_tables</td>
<td>OFF</td>
</tr>
<tr>
<td>ndb_slave_conflict_role</td>
<td>NONE</td>
</tr>
<tr>
<td>ndb_table_no_logging</td>
<td>OFF</td>
</tr>
<tr>
<td>ndb_table_temporary</td>
<td>OFF</td>
</tr>
<tr>
<td>ndb_use_copying_alter_table</td>
<td>OFF</td>
</tr>
<tr>
<td>ndb_use_exact_count</td>
<td>OFF</td>
</tr>
<tr>
<td>ndb_use_transactions</td>
<td>ON</td>
</tr>
<tr>
<td>ndb_version</td>
<td>524308</td>
</tr>
<tr>
<td>ndb_version_string</td>
<td>ndb-8.0.24</td>
</tr>
<tr>
<td>ndb_wait_connected</td>
<td>30</td>
</tr>
<tr>
<td>ndb_wait_setup</td>
<td>30</td>
</tr>
<tr>
<td>ndbinfo_database</td>
<td>ndbinfo</td>
</tr>
<tr>
<td>ndbinfo_max_bytes</td>
<td>0</td>
</tr>
<tr>
<td>ndbinfo_max_rows</td>
<td>10</td>
</tr>
<tr>
<td>ndbinfo_offline</td>
<td>OFF</td>
</tr>
<tr>
<td>ndbinfo_show_hidden</td>
<td>OFF</td>
</tr>
<tr>
<td>ndbinfo_table_prefix</td>
<td>ndb$</td>
</tr>
<tr>
<td>ndbinfo_version</td>
<td>524308</td>
</tr>
</tbody>
</table>

See Server System Variables, for more information.

- SELECT * FROM performance_schema.global_variables WHERE VARIABLE_NAME LIKE 'NDB%'

This statement is the equivalent of the SHOW VARIABLES statement described in the previous item, and provides almost identical output, as shown here:

```
mysql> SELECT * FROM performance_schema.global_variables
    ->  WHERE VARIABLE_NAME LIKE 'NDB%';
+--------------------------------------+---------------------------------------+
| VARIABLE_NAME                        | VARIABLE_VALUE                        |
+--------------------------------------+---------------------------------------+
| ndb_allow_copying_alter_table        | ON                                    |
| ndb_autoincrement_prefetch_sz        | 512                                   |
| ndb_batch_size                       | 32768                                 |
| ndb_blob_read_batch_bytes            | 65536                                 |
| ndb_blob_write_batch_bytes           | 65536                                 |
| ndb_clear_apply_status               | ON                                    |
| ndb_cluster_connection_pool          | 1                                     |
| ndb_cluster_connection_pool_nodeids  |                                       |
| ndb_connectstring                    | 127.0.0.1                             |
| ndb_data_node_neighbour              | 0                                     |
| ndb_default_column_format            | FIXED                                 |
| ndb_deferred_constraints             | 0                                     |
| ndb_distribution                     | KEYHASH                               |
| ndb_eventbuffer_free_percent         | 20                                    |
| ndb_eventbuffer_max_alloc            | 0                                     |
| ndb_extra_logging                    | 1                                     |
| ndb_force_send                       | ON                                    |
| ndb_fully_replicated                 | OFF                                   |
| ndb_index_stat_enable                | ON                                    |
| ndb_index_stat_option                | loop_enable=1000ms,loop_idle=1000ms, loop_busy=100ms,update_batch=1,read_batch=4, idle_batch=32,check_batch=8, check_delay=10m,delete_batch=8,clean_delay=1m, error_batch=4,error_delay=1m, evict_batch=8,evict_delay=1m,cache_limit=32M,cache_lowpct=90,zero_total=0 |
| ndb_join_pushdown                    | ON                                    |
| ndb_log_apply_stat                   | OFF                                   |
| ndb_log_bin                          | OFF                                   |
| ndb_log_binlog_index                 | ON                                    |
| ndb_log_empty_epochs                 | OFF                                   |
+--------------------------------------+---------------------------------------+
```

523
Unlike the case with the `SHOW VARIABLES` statement, it is possible to select individual columns. For example:

```sql
mysql> SELECT VARIABLE_NAME AS Name, VARIABLE_VALUE AS Value
    > FROM performance_schema.global_variables
    > WHERE VARIABLE_NAME
    >     IN ('version', 'ndb_version',
    >          'ndb_version_string', 'ndbinfo_version');
```

A more useful query is shown here:

```sql
mysql> SELECT VARIABLE_NAME AS Name, VARIABLE_VALUE AS Value
    > FROM performance_schema.global_variables
    > WHERE VARIABLE_NAME
    >     IN ('version', 'ndb_version',
    >          'ndb_version_string', 'ndbinfo_version');
```

```plaintext
+-----------------+-------------------+
| Name            | Value             |
|-----------------+-------------------|
| ndb_version     | 524308            |
| ndb_version_string | ndb-8.0.24     |
| ndbinfo_version | 524308            |
| version         | 8.0.20-cluster    |
```
For more information, see Performance Schema Status Variable Tables, and Server System Variables.

- **SHOW STATUS LIKE 'NDB%'**

This statement shows at a glance whether or not the MySQL server is acting as a cluster SQL node, and if so, it provides the MySQL server's cluster node ID, the host name and port for the cluster management server to which it is connected, and the number of data nodes in the cluster, as shown here:

```sql
mysql> SHOW STATUS LIKE 'NDB%';
+----------------------------------------------+-------------------------------+
| Variable_name                                | Value                         |
+----------------------------------------------+-------------------------------+
| Ndb_metadata_detected_count                  | 0                             |
| Ndb_cluster_node_id                          | 100                           |
| Ndb_config_from_host                         | 127.0.0.1                     |
| Ndb_config_from_port                         | 1186                          |
| Ndb_number_of_data_nodes                     | 2                             |
| Ndb_number_of_ready_data_nodes               | 2                             |
| Ndb_connect_count                            | 0                             |
| Ndb_execute_count                            | 0                             |
| Ndb_scan_count                               | 0                             |
| Ndb_pruned_scan_count                        | 0                             |
| Ndb_schema_locks_count                       | 0                             |
| Ndb_api_wait_exec_complete_count_session     | 0                             |
| Ndb_api_wait_scan_result_count_session       | 0                             |
| Ndb_api_wait_meta_request_count_session      | 1                             |
| Ndb_api_wait_nanos_count_session             | 163446                        |
| Ndb_api_bytes_sent_count_session             | 60                            |
| Ndb_api_bytes_received_count_session         | 28                            |
| Ndb_api_trans_start_count_session            | 0                             |
| Ndb_api_trans_commit_count_session           | 0                             |
| Ndb_api_trans_abort_count_session            | 0                             |
| Ndb_api_trans_close_count_session            | 0                             |
| Ndb_api_pk_op_count_session                  | 0                             |
| Ndb_api_uk_op_count_session                  | 0                             |
| Ndb_api_table_scan_count_session             | 0                             |
| Ndb_api_range_scan_count_session             | 0                             |
| Ndb_api_pruned_scan_count_session            | 0                             |
| Ndb_api_scan_batch_count_session             | 0                             |
| Ndb_api_read_row_count_session               | 0                             |
| Ndb_api_trans_local_read_row_count_session   | 0                             |
| Ndb_api_adaptive_send_forced_count_session   | 0                             |
| Ndb_api_adaptive_send_unforced_count_session | 0                             |
| Ndb_api_adaptive_send_deferred_count_session | 0                             |
| Ndb_api_trans_hint_count_session             | 0                             |
| Ndb_sorted_scan_count                        | 0                             |
| Ndb_pushed_queries_defined                   | 0                             |
| Ndb_pushed_queries_dropped                   | 0                             |
| Ndb_pushed_queries_executed                  | 0                             |
| Ndb_last_commit_epoch_server                 | 37632503447571                |
| Ndb_last_commit_epoch_session                | 0                             |
| Ndb_system_name                              | MC_20191126162038             |
+----------------------------------------------+-------------------------------+
"
If the MySQL server was built with NDB support, but it is not currently connected to a cluster, every row in the output of this statement contains a zero or an empty string for the Value column.

See also SHOW STATUS Statement.
NDB Cluster Security Issues

- SELECT * FROM performance_schema.global_status WHERE VARIABLE_NAME LIKE 'NDB%'

This statement provides similar output to the SHOW STATUS statement discussed in the previous item. Unlike the case with SHOW STATUS, it is possible using SELECT statements to extract values in SQL for use in scripts for monitoring and automation purposes.

For more information, see Performance Schema Status Variable Tables.

- SELECT * FROM INFORMATION_SCHEMA.PLUGINS WHERE PLUGIN_NAME LIKE 'NDB%'

This statement displays information from the INFORMATION_SCHEMA.PLUGINS table about plugins associated with NDB Cluster, such as version, author, and license, as shown here:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.PLUGINS
-> WHERE PLUGIN_NAME LIKE 'NDB%'
G
*************************** 1. row ***************************
PLUGIN_NAME: ndbcluster
PLUGIN_VERSION: 1.0
PLUGIN_STATUS: ACTIVE
PLUGIN_TYPE: STORAGE ENGINE
PLUGIN_TYPE_VERSION: 80025.0
PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
PLUGIN_AUTHOR: Oracle Corporation
PLUGIN_DESCRIPTION: Clustered, fault-tolerant tables
PLUGIN_LICENSE: GPL
LOAD_OPTION: ON
*************************** 2. row ***************************
PLUGIN_NAME: ndbinfo
PLUGIN_VERSION: 0.1
PLUGIN_STATUS: ACTIVE
PLUGIN_TYPE: STORAGE ENGINE
PLUGIN_TYPE_VERSION: 80025.0
PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
PLUGIN_AUTHOR: Oracle Corporation
PLUGIN_DESCRIPTION: MySQL Cluster system information storage engine
PLUGIN_LICENSE: GPL
LOAD_OPTION: ON
*************************** 3. row ***************************
PLUGIN_NAME: ndb_transid_mysql_connection_map
PLUGIN_VERSION: 0.1
PLUGIN_STATUS: ACTIVE
PLUGIN_TYPE: INFORMATION_SCHEMA
PLUGIN_TYPE_VERSION: 80025.0
PLUGIN_LIBRARY: NULL
PLUGIN_LIBRARY_VERSION: NULL
PLUGIN_AUTHOR: Oracle Corporation
PLUGIN_DESCRIPTION: Map between MySQL connection ID and NDB transaction ID
PLUGIN_LICENSE: GPL
LOAD_OPTION: ON
```

You can also use the SHOW PLUGINS statement to display this information, but the output from that statement cannot easily be filtered. See also The MySQL Plugin API, which describes where and how the information in the PLUGINS table is obtained.

You can also query the tables in the ndbinfo information database for real-time data about many NDB Cluster operations. See Section 7.14, “ndbinfo: The NDB Cluster Information Database”.

7.17 NDB Cluster Security Issues

This section discusses security considerations to take into account when setting up and running NDB Cluster.

Topics covered in this section include the following:
• NDB Cluster and network security issues
• Configuration issues relating to running NDB Cluster securely
• NDB Cluster and the MySQL privilege system
• MySQL standard security procedures as applicable to NDB Cluster

7.17.1 NDB Cluster Security and Networking Issues

In this section, we discuss basic network security issues as they relate to NDB Cluster. It is extremely important to remember that NDB Cluster “out of the box” is not secure; you or your network administrator must take the proper steps to ensure that your cluster cannot be compromised over the network.

Cluster communication protocols are inherently insecure, and no encryption or similar security measures are used in communications between nodes in the cluster. Because network speed and latency have a direct impact on the cluster's efficiency, it is also not advisable to employ SSL or other encryption to network connections between nodes, as such schemes cause slow communications.

It is also true that no authentication is used for controlling API node access to an NDB Cluster. As with encryption, the overhead of imposing authentication requirements would have an adverse impact on Cluster performance.

In addition, there is no checking of the source IP address for either of the following when accessing the cluster:

• SQL or API nodes using “free slots” created by empty [mysqld] or [api] sections in the config.ini file

This means that, if there are any empty [mysqld] or [api] sections in the config.ini file, then any API nodes (including SQL nodes) that know the management server's host name (or IP address) and port can connect to the cluster and access its data without restriction. (See Section 7.17.2, “NDB Cluster and MySQL Privileges”, for more information about this and related issues.)

Note

You can exercise some control over SQL and API node access to the cluster by specifying a HostName parameter for all [mysqld] and [api] sections in the config.ini file. However, this also means that, should you wish to connect an API node to the cluster from a previously unused host, you need to add an [api] section containing its host name to the config.ini file.

More information is available elsewhere in this chapter about the HostName parameter. Also see Section 5.1, “Quick Test Setup of NDB Cluster”, for configuration examples using HostName with API nodes.

• Any ndb_mgm client

This means that any cluster management client that is given the management server's host name (or IP address) and port (if not the standard port) can connect to the cluster and execute any management client command. This includes commands such as ALL STOP and SHUTDOWN.

For these reasons, it is necessary to protect the cluster on the network level. The safest network configuration for Cluster is one which isolates connections between Cluster nodes from any other network communications. This can be accomplished by any of the following methods:

1. Keeping Cluster nodes on a network that is physically separate from any public networks. This option is the most dependable; however, it is the most expensive to implement.

   We show an example of an NDB Cluster setup using such a physically segregated network here:
This setup has two networks, one private (solid box) for the Cluster management servers and data nodes, and one public (dotted box) where the SQL nodes reside. (We show the management and data nodes connected using a gigabit switch since this provides the best performance.) Both networks are protected from the outside by a hardware firewall, sometimes also known as a network-based firewall.

This network setup is safest because no packets can reach the cluster's management or data nodes from outside the network—and none of the cluster's internal communications can reach the outside—without going through the SQL nodes, as long as the SQL nodes do not permit any packets to be forwarded. This means, of course, that all SQL nodes must be secured against hacking attempts.

Important

With regard to potential security vulnerabilities, an SQL node is no different from any other MySQL server. See Making MySQL Secure Against Attackers, for a description of techniques you can use to secure MySQL servers.

2. Using one or more software firewalls (also known as host-based firewalls) to control which packets pass through to the cluster from portions of the network that do not require access to it. In this type
of setup, a software firewall must be installed on every host in the cluster which might otherwise be accessible from outside the local network.

The host-based option is the least expensive to implement, but relies purely on software to provide protection and so is the most difficult to keep secure.

This type of network setup for NDB Cluster is illustrated here:

**Figure 7.3 NDB Cluster with Software Firewalls**

Using this type of network setup means that there are two zones of NDB Cluster hosts. Each cluster host must be able to communicate with all of the other machines in the cluster, but only those hosting SQL nodes (dotted box) can be permitted to have any contact with the outside, while those in the zone containing the data nodes and management nodes (solid box) must be isolated from any machines that are not part of the cluster. Applications using the cluster and user of those applications must not be permitted to have direct access to the management and data node hosts.

To accomplish this, you must set up software firewalls that limit the traffic to the type or types shown in the following table, according to the type of node that is running on each cluster host computer:

**Table 7.16 Node types in a host-based firewall cluster configuration**

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Permitted Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL or API node</td>
<td>• It originates from the IP address of a management or data node (using any TCP or UDP port).</td>
</tr>
<tr>
<td></td>
<td>• It originates from within the network in which the cluster resides and is on the port that your application is using.</td>
</tr>
<tr>
<td>Data node or Management node</td>
<td>• It originates from the IP address of a management or data node (using any TCP or UDP port).</td>
</tr>
</tbody>
</table>
### NDB Cluster Security and Networking Issues

<table>
<thead>
<tr>
<th>Node Type</th>
<th>Permitted Traffic</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>• It originates from the IP address of an SQL or API node.</td>
</tr>
</tbody>
</table>

Any traffic other than that shown in the table for a given node type should be denied.

The specifics of configuring a firewall vary from firewall application to firewall application, and are beyond the scope of this Manual. **iptables** is a very common and reliable firewall application, which is often used with **APF** as a front end to make configuration easier. You can (and should) consult the documentation for the software firewall that you employ, should you choose to implement an NDB Cluster network setup of this type, or of a “mixed” type as discussed under the next item.

3. It is also possible to employ a combination of the first two methods, using both hardware and software to secure the cluster—that is, using both network-based and host-based firewalls. This is between the first two schemes in terms of both security level and cost. This type of network setup keeps the cluster behind the hardware firewall, but permits incoming packets to travel beyond the router connecting all cluster hosts to reach the SQL nodes.

One possible network deployment of an NDB Cluster using hardware and software firewalls in combination is shown here:

**Figure 7.4 NDB Cluster with a Combination of Hardware and Software Firewalls**

In this case, you can set the rules in the hardware firewall to deny any external traffic except to SQL nodes and API nodes, and then permit traffic to them only on the ports required by your application.

Whatever network configuration you use, remember that your objective from the viewpoint of keeping the cluster secure remains the same—to prevent any unessential traffic from reaching the cluster while ensuring the most efficient communication between the nodes in the cluster.

Because NDB Cluster requires large numbers of ports to be open for communications between nodes, the recommended option is to use a segregated network. This represents the simplest way to prevent unwanted traffic from reaching the cluster.

---

**Note**

If you wish to administer an NDB Cluster remotely (that is, from outside the local network), the recommended way to do this is to use **ssh** or another secure login shell to access an SQL node host. From this host, you can then run the
management client to access the management server safely, from within the cluster's own local network.

Even though it is possible to do so in theory, it is not recommended to use ndb_mgm to manage a Cluster directly from outside the local network on which the Cluster is running. Since neither authentication nor encryption takes place between the management client and the management server, this represents an extremely insecure means of managing the cluster, and is almost certain to be compromised sooner or later.

7.17.2 NDB Cluster and MySQL Privileges

In this section, we discuss how the MySQL privilege system works in relation to NDB Cluster and the implications of this for keeping an NDB Cluster secure.

Standard MySQL privileges apply to NDB Cluster tables. This includes all MySQL privilege types (SELECT privilege, UPDATE privilege, DELETE privilege, and so on) granted on the database, table, and column level. As with any other MySQL Server, user and privilege information is stored in the mysql system database. The SQL statements used to grant and revoke privileges on NDB tables, databases containing such tables, and columns within such tables are identical in all respects with the GRANT and REVOKE statements used in connection with database objects involving any (other) MySQL storage engine. The same thing is true with respect to the CREATE USER and DROP USER statements.

It is important to keep in mind that, by default, the MySQL grant tables use the MyISAM storage engine. Because of this, those tables are not normally duplicated or shared among MySQL servers acting as SQL nodes in an NDB Cluster. In other words, changes in users and their privileges do not automatically propagate between SQL nodes by default. If you wish, you can enable automatic distribution of MySQL users and privileges across NDB Cluster SQL nodes; see Section 7.12, “Distributed MySQL Privileges with NDB_STORED_USER”, for details.

Conversely, because there is no way in MySQL to deny privileges (privileges can either be revoked or not granted in the first place, but not denied as such), there is no special protection for NDB tables on one SQL node from users that have privileges on another SQL node; (This is true even if you are not using automatic distribution of user privileges. The definitive example of this is the MySQL root account, which can perform any action on any database object. In combination with empty [mysqld] or [api] sections of the config.ini file, this account can be especially dangerous. To understand why, consider the following scenario:

- The config.ini file contains at least one empty [mysqld] or [api] section. This means that the NDB Cluster management server performs no checking of the host from which a MySQL Server (or other API node) accesses the NDB Cluster.
- There is no firewall, or the firewall fails to protect against access to the NDB Cluster from hosts external to the network.
- The host name or IP address of the NDB Cluster management server is known or can be determined from outside the network.

If these conditions are true, then anyone, anywhere can start a MySQL Server with --ndbcluster --ndb-connectstring=management_host and access this NDB Cluster. Using the MySQL root account, this person can then perform the following actions:

- Execute metadata statements such as SHOW DATABASES statement (to obtain a list of all NDB databases on the server) or SHOW TABLES FROM some_ndb_database statement to obtain a list of all NDB tables in a given database
- Run any legal MySQL statements on any of the discovered tables, such as:
  - SELECT * FROM some_table to read all the data from any table
  - DELETE FROM some_table to delete all the data from a table
NDB Cluster and MySQL Security Procedures

- **DESCRIBE some_table** or **SHOW CREATE TABLE some_table** to determine the table schema.

- **UPDATE some_table SET column1 = some_value** to fill a table column with "garbage" data; this could actually cause much greater damage than simply deleting all the data.

More insidious variations might include statements like these:

```sql
UPDATE some_table SET an_int_column = an_int_column + 1
```

or

```sql
UPDATE some_table SET a_varchar_column = REVERSE(a_varchar_column)
```

Such malicious statements are limited only by the imagination of the attacker.

The only tables that would be safe from this sort of mayhem would be those tables that were created using storage engines other than NDB, and so not visible to a "rogue" SQL node.

A user who can log in as `root` can also access the `INFORMATION_SCHEMA` database and its tables, and so obtain information about databases, tables, stored routines, scheduled events, and any other database objects for which metadata is stored in `INFORMATION_SCHEMA`.

It is also a very good idea to use different passwords for the `root` accounts on different NDB Cluster SQL nodes unless you are using distributed privileges.

In sum, you cannot have a safe NDB Cluster if it is directly accessible from outside your local network.

Important

*Never leave the MySQL root account password empty.* This is just as true when running MySQL as an NDB Cluster SQL node as it is when running it as a standalone (non-Cluster) MySQL Server, and should be done as part of the MySQL installation process before configuring the MySQL Server as an SQL node in an NDB Cluster.

If you wish to employ NDB Cluster's distributed privilege capabilities, you should not simply convert the system tables in the `mysql` database to use the NDB storage engine manually. Use the stored procedure provided for this purpose instead; see Section 7.12, "Distributed MySQL Privileges with NDB_STORED_USER".

Otherwise, if you need to synchronize `mysql` system tables between SQL nodes, you can use standard MySQL replication to do so, or employ a script to copy table entries between the MySQL servers.

**Summary.** The most important points to remember regarding the MySQL privilege system with regard to NDB Cluster are listed here:

1. Users and privileges established on one SQL node do not automatically exist or take effect on other SQL nodes in the cluster. Conversely, removing a user or privilege on one SQL node in the cluster does not remove the user or privilege from any other SQL nodes.

2. You can distribute MySQL users and privileges among SQL nodes using the SQL script, and the stored procedures it contains, that are supplied for this purpose in the NDB Cluster distribution.

3. Once a MySQL user is granted privileges on an NDB table from one SQL node in an NDB Cluster, that user can "see" any data in that table regardless of the SQL node from which the data originated, even if you are not using privilege distribution.

**7.17.3 NDB Cluster and MySQL Security Procedures**

In this section, we discuss MySQL standard security procedures as they apply to running NDB Cluster.
**NDB Cluster and MySQL Security Procedures**

In general, any standard procedure for running MySQL securely also applies to running a MySQL Server as part of an NDB Cluster. First and foremost, you should always run a MySQL Server as the `mysql` operating system user; this is no different from running MySQL in a standard (non-Cluster) environment. The `mysql` system account should be uniquely and clearly defined. Fortunately, this is the default behavior for a new MySQL installation. You can verify that the `mysqld` process is running as the `mysql` operating system user by using the system command such as the one shown here:

```
shell> ps aux | grep mysql
```

```
root     10467  0.0  0.1   3616  1380 pts/3    S    11:53   0:00 
        /bin/sh ./mysqld_safe --ndbcluster --ndb-connectstring=localhost:1186
mysql    10512  0.2  2.5  58528 26636 pts/3    S  11:53   0:00 
        /usr/local/mysql/libexec/mysqld --basedir=/usr/local/mysql 
        --datadir=/usr/local/mysql/var --user=mysql --ndbcluster 
        --ndb-connectstring=localhost:1186 --pid-file=/usr/local/mysql/var/mothra.pid 
        --log-error=/usr/local/mysql/var/mothra.err
jon      10579  0.0  0.0   2736   688 pts/0    S+   11:54   0:00 grep mysql
```

If the `mysqld` process is running as any other user than `mysql`, you should immediately shut it down and restart it as the `mysql` user. If this user does not exist on the system, the `mysql` user account should be created, and this user should be part of the `mysql` user group; in this case, you should also make sure that the MySQL data directory on this system (as set using the `--datadir` option for `mysqld`) is owned by the `mysql` user, and that the SQL node's `my.cnf` file includes `user=mysql` in the `[mysqld]` section. Alternatively, you can start the MySQL server process with `--user=mysql` on the command line, but it is preferable to use the `my.cnf` option, since you might forget to use the command-line option and so have `mysqld` running as another user unintentionally. The `mysqld_safe` startup script forces MySQL to run as the `mysql` user.

**Important**

Never run `mysqld` as the system root user. Doing so means that potentially any file on the system can be read by MySQL, and thus—should MySQL be compromised—by an attacker.

As mentioned in the previous section (see Section 7.17.2, “NDB Cluster and MySQL Privileges”), you should always set a root password for the MySQL Server as soon as you have it running. You should also delete the anonymous user account that is installed by default. You can accomplish these tasks using the following statements:

```
shell> mysql -u root
mysql> UPDATE mysql.user 
    -->  SET Password=PASSWORD('secure_password')
    -->  WHERE User='root';
mysql> DELETE FROM mysql.user 
    -->  WHERE User='';
mysql> FLUSH PRIVILEGES;
```

Be very careful when executing the `DELETE` statement not to omit the `WHERE` clause, or you risk deleting all MySQL users. Be sure to run the `FLUSH PRIVILEGES` statement as soon as you have modified the `mysql.user` table, so that the changes take immediate effect. Without `FLUSH PRIVILEGES`, the changes do not take effect until the next time that the server is restarted.

**Note**

Many of the NDB Cluster utilities such as `ndb_show_tables`, `ndb_desc`, and `ndb_select_all` also work without authentication and can reveal table names, schemas, and data. By default these are installed on Unix-style systems with the permissions `wxr-xr-x` (755), which means they can be executed by any user that can access the `mysql/bin` directory.

See Chapter 6, *NDB Cluster Programs*, for more information about these utilities.
Chapter 8 NDB Cluster Replication

Table of Contents

8.1 NDB Cluster Replication: Abbreviations and Symbols ................................................................. 536
8.2 General Requirements for NDB Cluster Replication ................................................................. 537
8.3 Known Issues in NDB Cluster Replication ............................................................................... 538
8.4 NDB Cluster Replication Schema and Tables ........................................................................... 545
8.5 Preparing the NDB Cluster for Replication .............................................................................. 547
8.6 Starting NDB Cluster Replication (Single Replication Channel) ............................................. 549
8.7 Using Two Replication Channels for NDB Cluster Replication ................................................ 551
8.8 Implementing Failover with NDB Cluster Replication ............................................................. 552
8.9 NDB Cluster Backups With NDB Cluster Replication ............................................................. 554
  8.9.1 NDB Cluster Replication: Automating Synchronization of the Replica to the Source Binary Log .................................................................................................................. 556
  8.9.2 Point-In-Time Recovery Using NDB Cluster Replication ...................................................... 558
8.10 NDB Cluster Replication: Bidirectional and Circular Replication ........................................... 559
8.11 NDB Cluster Replication Conflict Resolution .......................................................................... 563

NDB Cluster supports asynchronous replication, more usually referred to simply as “replication”. This section explains how to set up and manage a configuration in which one group of computers operating as an NDB Cluster replicates to a second computer or group of computers. We assume some familiarity on the part of the reader with standard MySQL replication as discussed elsewhere in this Manual. (See Replication).

Note

NDB Cluster does not support replication using GTIDs; semisynchronous replication and group replication are also not supported by the NDB storage engine.

Normal (non-clustered) replication involves a source server and a replica server, the source being so named because operations and data to be replicated originate with it, and the replica being the recipient of these. In NDB Cluster, replication is conceptually very similar but can be more complex in practice, as it may be extended to cover a number of different configurations including replicating between two complete clusters. Although an NDB Cluster itself depends on the NDB storage engine for clustering functionality, it is not necessary to use NDB as the storage engine for the replica’s copies of the replicated tables (see Replication from NDB to other storage engines). However, for maximum availability, it is possible (and preferable) to replicate from one NDB Cluster to another, and it is this scenario that we discuss, as shown in the following figure:
In this scenario, the replication process is one in which successive states of a source cluster are logged and saved to a replica cluster. This process is accomplished by a special thread known as the NDB binary log injector thread, which runs on each MySQL server and produces a binary log (binlog). This thread ensures that all changes in the cluster producing the binary log—and not just those changes that are effected through the MySQL Server—are inserted into the binary log with the correct serialization order. We refer to the MySQL source and replica servers as replication servers or replication nodes, and the data flow or line of communication between them as a replication channel.

For information about performing point-in-time recovery with NDB Cluster and NDB Cluster Replication, see Section 8.9.2, “Point-In-Time Recovery Using NDB Cluster Replication”.

**NDB API replica status variables.** NDB API counters can provide enhanced monitoring capabilities on replica clusters. These counters are implemented as NDB statistics _slave status variables, as seen in the output of SHOW STATUS, or in the results of queries against the Performance Schema session_status or global_status table in a mysql client session connected to a MySQL Server that is acting as a replica in NDB Cluster Replication. By comparing the values of these status variables before and after the execution of statements affecting replicated NDB tables, you can observe the corresponding actions taken on the NDB API level by the replica, which can be useful when monitoring or troubleshooting NDB Cluster Replication. Section 7.13, “NDB API Statistics Counters and Variables”, provides additional information.

**Replication from NDB to non-NDB tables.** It is possible to replicate NDB tables from an NDB Cluster acting as the replication source to tables using other MySQL storage engines such as InnoDB or MyISAM on a replica mysql. This is subject to a number of conditions; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

### 8.1 NDB Cluster Replication: Abbreviations and Symbols

Throughout this section, we use the following abbreviations or symbols for referring to the source and replica clusters, and to processes and commands run on the clusters or cluster nodes:
### Table 8.1 Abbreviations used throughout this section referring to source and replica clusters, and to processes and commands run on cluster nodes

<table>
<thead>
<tr>
<th>Symbol or Abbreviation</th>
<th>Description (Refers to...)</th>
</tr>
</thead>
<tbody>
<tr>
<td>S</td>
<td>The cluster serving as the (primary) replication source</td>
</tr>
<tr>
<td>R</td>
<td>The cluster acting as the (primary) replica</td>
</tr>
<tr>
<td>shellS&gt;</td>
<td>Shell command to be issued on the source cluster</td>
</tr>
<tr>
<td>mysqlS&gt;</td>
<td>MySQL client command issued on a single MySQL server running as an SQL node on the source cluster</td>
</tr>
<tr>
<td>mysqlS*&gt;</td>
<td>MySQL client command to be issued on all SQL nodes participating in the replication source cluster</td>
</tr>
<tr>
<td>shellR&gt;</td>
<td>Shell command to be issued on the replica cluster</td>
</tr>
<tr>
<td>mysqlR&gt;</td>
<td>MySQL client command issued on a single MySQL server running as an SQL node on the replica cluster</td>
</tr>
<tr>
<td>mysqlR*&gt;</td>
<td>MySQL client command to be issued on all SQL nodes participating in the replica cluster</td>
</tr>
<tr>
<td>C</td>
<td>Primary replication channel</td>
</tr>
<tr>
<td>C’</td>
<td>Secondary replication channel</td>
</tr>
<tr>
<td>S’</td>
<td>Secondary replication source</td>
</tr>
<tr>
<td>R’</td>
<td>Secondary replica</td>
</tr>
</tbody>
</table>

### 8.2 General Requirements for NDB Cluster Replication

A replication channel requires two MySQL servers acting as replication servers (one each for the source and replica). For example, this means that in the case of a replication setup with two replication channels (to provide an extra channel for redundancy), there are a total of four replication nodes, two per cluster.

Replication of an NDB Cluster as described in this section and those following is dependent on row-based replication. This means that the replication source MySQL server must be running with `--binlog-format=ROW` or `--binlog-format=MIXED`, as described in Section 8.6, “Starting NDB Cluster Replication (Single Replication Channel)”. For general information about row-based replication, see Replication Formats.

**Important**

If you attempt to use NDB Cluster Replication with `--binlog-format=STATEMENT`, replication fails to work properly because the `ndb_binlog_index` table on the source cluster and the `epoch` column of the `ndb_apply_status` table on the replica cluster are not updated (see Section 8.4, “NDB Cluster Replication Schema and Tables”). Instead, only updates on the MySQL server acting as the replication source propagate to the replica, and no updates from any other SQL nodes in the source cluster are replicated.

The default value for the `--binlog-format` option is `MIXED`.

Each MySQL server used for replication in either cluster must be uniquely identified among all the MySQL replication servers participating in either cluster (you cannot have replication servers on both the source and replica clusters sharing the same ID). This can be done by starting each SQL node using the `--server-id=id` option, where `id` is a unique integer. Although it is not strictly necessary, we assume for purposes of this discussion that all NDB Cluster binaries are of the same release version.
It is generally true in MySQL Replication that both MySQL servers (mysqld processes) involved must be compatible with one another with respect to both the version of the replication protocol used and the SQL feature sets which they support (see Replication Compatibility Between MySQL Versions). It is due to such differences between the binaries in the NDB Cluster and MySQL Server 8.0 distributions that NDB Cluster Replication has the additional requirement that both mysqld binaries come from an NDB Cluster distribution. The simplest and easiest way to assure that the mysqld servers are compatible is to use the same NDB Cluster distribution for all source and replica mysqld binaries.

We assume that the replica server or cluster is dedicated to replication of the source cluster, and that no other data is being stored on it.

All NDB tables being replicated must be created using a MySQL server and client. Tables and other database objects created using the NDB API (with, for example, Dictionary::createTable()) are not visible to a MySQL server and so are not replicated. Updates by NDB API applications to existing tables that were created using a MySQL server can be replicated.

Note

It is possible to replicate an NDB Cluster using statement-based replication. However, in this case, the following restrictions apply:

- All updates to data rows on the cluster acting as the source must be directed to a single MySQL server.
- It is not possible to replicate a cluster using multiple simultaneous MySQL replication processes.
- Only changes made at the SQL level are replicated.

These are in addition to the other limitations of statement-based replication as opposed to row-based replication; see Advantages and Disadvantages of Statement-Based and Row-Based Replication, for more specific information concerning the differences between the two replication formats.

8.3 Known Issues in NDB Cluster Replication

This section discusses known problems or issues when using replication with NDB Cluster.

Loss of connection between source and replica. A loss of connection can occur either between the source cluster SQL node and the replica cluster SQL node, or between the source SQL node and the data nodes of the source cluster. In the latter case, this can occur not only as a result of loss of physical connection (for example, a broken network cable), but due to the overflow of data node event buffers; if the SQL node is too slow to respond, it may be dropped by the cluster (this is controllable to some degree by adjusting the MaxBufferedEpochs and TimeBetweenEpochs configuration parameters). If this occurs, it is entirely possible for new data to be inserted into the source cluster without being recorded in the source SQL node's binary log. For this reason, to guarantee high availability, it is extremely important to maintain a backup replication channel, to monitor the primary channel, and to fail over to the secondary replication channel when necessary to keep the replica cluster synchronized with the source. NDB Cluster is not designed to perform such monitoring on its own; for this, an external application is required.

The source SQL node issues a “gap” event when connecting or reconnecting to the source cluster. (A gap event is a type of “incident event,” which indicates an incident that occurs which affects the contents of the database but that cannot easily be represented as a set of changes. Examples of incidents are server failures, database resynchronization, some software updates, and some hardware changes.) When the replica encounters a gap in the replication log, it stops with an error message. This message is available in the output of SHOW REPLICA | SLAVE STATUS, and indicates that the SQL thread has stopped due to an incident registered in the replication stream, and that manual intervention is required. See Section 8.8, “Implementing Failover with NDB Cluster Replication”, for more information about what to do in such circumstances.
Important

Because NDB Cluster is not designed on its own to monitor replication status or provide failover, if high availability is a requirement for the replica server or cluster, then you must set up multiple replication lines, monitor the source mysqld on the primary replication line, and be prepared fail over to a secondary line if and as necessary. This must be done manually, or possibly by means of a third-party application. For information about implementing this type of setup, see Section 8.7, “Using Two Replication Channels for NDB Cluster Replication”, and Section 8.8, “Implementing Failover with NDB Cluster Replication”.

If you are replicating from a standalone MySQL server to an NDB Cluster, one channel is usually sufficient.

Circular replication. NDB Cluster Replication supports circular replication, as shown in the next example. The replication setup involves three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication source for Cluster 2, Cluster 2 acts as the source for Cluster 3, and Cluster 3 acts as the source for Cluster 1, thus completing the circle. Each NDB Cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all source and replica clusters are the same.
- All SQL nodes acting as sources and replicas are started with the log_slave_updates system variable enabled.

This type of circular replication setup is shown in the following diagram:
In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the curved arrows in the diagram) directly connects all SQL nodes used as sources and replicas.

It should also be possible to set up circular replication in which not all source SQL nodes are also replicas, as shown here:
Known Issues in NDB Cluster Replication

In this case, different SQL nodes in each cluster are used as sources and replicas. However, you must not start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

**Note**

The NDB storage engine uses idempotent execution mode, which suppresses duplicate-key and other errors that otherwise break circular replication of NDB Cluster. This is equivalent to setting the global `slave_exec_mode` system variable to `IDEMPOTENT`, although this is not necessary in NDB Cluster replication, since NDB Cluster sets this variable automatically and ignores any attempts to set it explicitly.

**NDB Cluster replication and primary keys.** In the event of a node failure, errors in replication of NDB tables without primary keys can still occur, due to the possibility of duplicate rows being inserted in such cases. For this reason, it is highly recommended that all NDB tables being replicated have explicit primary keys.

**NDB Cluster Replication and Unique Keys.** In older versions of NDB Cluster, operations that updated values of unique key columns of NDB tables could result in duplicate-key errors when replicated. This issue is solved for replication between NDB tables by deferring unique key checks until after all table row updates have been performed.
Deferring constraints in this way is currently supported only by NDB. Thus, updates of unique keys when replicating from NDB to a different storage engine such as InnoDB or MyISAM are still not supported.

The problem encountered when replicating without deferred checking of unique key updates can be illustrated using NDB table such as t, is created and populated on the source (and transmitted to a replica that does not support deferred unique key updates) as shown here:

```sql
CREATE TABLE t (
    p INT PRIMARY KEY,
    c INT,
    UNIQUE KEY u (c)
) ENGINE NDB;
INSERT INTO t
VALUES (1,1), (2,2), (3,3), (4,4), (5,5);
```

The following `UPDATE` statement on t succeeds on the source, since the rows affected are processed in the order determined by the `ORDER BY` option, performed over the entire table:

```sql
UPDATE t SET c = c - 1 ORDER BY p;
```

The same statement fails with a duplicate key error or other constraint violation on the replica, because the ordering of the row updates is performed for one partition at a time, rather than for the table as a whole.

**Note**

Every NDB table is implicitly partitioned by key when it is created. See KEY Partitioning, for more information.

**GTIDs not supported.** Replication using global transaction IDs is not compatible with the NDB storage engine, and is not supported. Enabling GTIDs is likely to cause NDB Cluster Replication to fail.

**Multithreaded replicas not supported.** NDB Cluster does not support multithreaded replicas, and setting related system variables such as `slave_parallel_workers`, `slave_checkpoint_group`, and `slave_checkpoint_group` (or the equivalent `mysqld` startup options) has no effect.

This is because the replica may not be able to separate transactions occurring in one database from those in another if they are written within the same epoch. In addition, every transaction handled by the NDB storage engine involves at least two databases—the target database and the `mysql` system database—due to the requirement for updating the `mysql.ndb_apply_status` table (see Section 8.4, “NDB Cluster Replication Schema and Tables”). This in turn breaks the requirement for multithreading that the transaction is specific to a given database.

**Restarting with --initial.** Restarting the cluster with the `--initial` option causes the sequence of GCI and epoch numbers to start over from 0. (This is generally true of NDB Cluster and not limited to replication scenarios involving Cluster.) The MySQL servers involved in replication should in this case be restarted. After this, you should use the `RESET MASTER` and `RESET REPLICA | SLAVE` statements to clear the invalid `ndb_binlog_index` and `ndb_apply_status` tables, respectively.

**Replication from NDB to other storage engines.** It is possible to replicate an NDB table on the source to a table using a different storage engine on the replica, taking into account the restrictions listed here:

- Multi-source and circular replication are not supported (tables on both the source and the replica must use the NDB storage engine for this to work).
- Using a storage engine which does not perform binary logging for tables on the replica requires special handling.
- Use of a nontransactional storage engine for tables on the replica also requires special handling.
- The source `mysqld` must be started with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`.

542
The next few paragraphs provide additional information about each of the issues just described.

**Multiple sources not supported when replicating NDB to other storage engines.** For replication from NDB to a different storage engine, the relationship between the two databases must be one-to-one. This means that bidirectional or circular replication is not supported between NDB Cluster and other storage engines.

In addition, it is not possible to configure more than one replication channel when replicating between NDB and a different storage engine. (An NDB Cluster database can simultaneously replicate to multiple NDB Cluster databases.) If the source uses NDB tables, it is still possible to have more than one MySQL Server maintain a binary log of all changes, but for the replica to change sources (fail over), the new source-replica relationship must be explicitly defined on the replica.

**Replicating NDB tables to a storage engine that does not perform binary logging.** If you attempt to replicate from an NDB Cluster to a replica that uses a storage engine that does not handle its own binary logging, the replication process aborts with the error `Binary logging not possible ... Statement cannot be written atomically since more than one engine involved and at least one engine is self-logging (Error 1595)`. It is possible to work around this issue in one of the following ways:

- **Turn off binary logging on the replica.** This can be accomplished by setting `sql_log_bin = 0`.

- **Change the storage engine used for the mysql.ndb_apply_status table.** Causing this table to use an engine that does not handle its own binary logging can also eliminate the conflict. This can be done by issuing a statement such as `ALTER TABLE mysql.ndb_apply_status ENGINE=MyISAM` on the replica. It is safe to do this when using a storage engine other than NDB on the replica, since you do not need to worry about keeping multiple replicas synchronized.

- **Filter out changes to the mysql.ndb_apply_status table on the replica.** This can be done by starting the replica with `--replicate-ignore-table=mysql.ndb_apply_status`. If you need for other tables to be ignored by replication, you might wish to use an appropriate `--replicate-wild-ignore-table` option instead.

**Important**

You should not disable replication or binary logging of `mysql.ndb_apply_status` or change the storage engine used for this table when replicating from one NDB Cluster to another. See Replication and binary log filtering rules with replication between NDB Clusters, for details.

**Replication from NDB to a nontransactional storage engine.** When replicating from NDB to a nontransactional storage engine such as MyISAM, you may encounter unnecessary duplicate key errors when replicating `INSERT ... ON DUPLICATE KEY UPDATE` statements. You can suppress these by using `--ndb-log-update-as-write=0`, which forces updates to be logged as writes, rather than as updates.

**Replication and binary log filtering rules with replication between NDB Clusters.** If you are using any of the options `--replicate-do-*,--replicate-ignore-*,--binlog-do-db, or --binlog-ignore-db` to filter databases or tables being replicated, you must take care not to block replication or binary logging of the `mysql.ndb_apply_status`, which is required for replication between NDB Clusters to operate properly. In particular, you must keep in mind the following:

1. Using `--replicate-do-db=db_name` (and no other `--replicate-do-*` or `--replicate-ignore-*` options) means that only tables in database `db_name` are replicated. In this case, you should also use `--replicate-do-db=mysql, --binlog-do-db=mysql, or --replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on replicas.

Using `--binlog-do-db=db_name` (and no other `--binlog-do-db` options) means that changes only to tables in database `db_name` are written to the binary log. In this case, you should
also use `--replicate-do-db=mysql, --binlog-do-db=mysql` or `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is populated on replicas.

2. Using `--replicate-ignore-db=mysql` means that no tables in the `mysql` database are replicated. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

Using `--binlog-ignore-db=mysql` means that no changes to tables in the `mysql` database are written to the binary log. In this case, you should also use `--replicate-do-table=mysql.ndb_apply_status` to ensure that `mysql.ndb_apply_status` is replicated.

You should also remember that each replication rule requires the following:

1. Its own `--replicate-do-*` or `--replicate-ignore-*` option, and that multiple rules cannot be expressed in a single replication filtering option. For information about these rules, see Replication and Binary Logging Options and Variables.

2. Its own `--binlog-do-db` or `--binlog-ignore-db` option, and that multiple rules cannot be expressed in a single binary log filtering option. For information about these rules, see The Binary Log.

If you are replicating an NDB Cluster to a replica that uses a storage engine other than NDB, the considerations just given previously may not apply, as discussed elsewhere in this section.

**NDB Cluster Replication and IPv6.** Beginning with NDB 8.0.22, all types of NDB Cluster nodes support IPv6; this includes management nodes, data nodes, and API or SQL nodes.

Prior to NDB 8.0.22, the NDB API and MGM API (and thus data nodes and management nodes) do not support IPv6, although MySQL Servers—including those acting as SQL nodes in an NDB Cluster—can use IPv6 to contact other MySQL Servers. In versions of NDB Cluster prior to 8.0.22, you can replicate between clusters using IPv6 to connect the SQL nodes acting as source and replica as shown by the dotted arrow in the following diagram:

**Figure 8.4 Replication Between SQL Nodes Connected Using IPv6**

Prior to NDB 8.0.22, all connections originating within the NDB Cluster—represented in the preceding diagram by solid arrows—must use IPv4. In other words, all NDB Cluster data nodes, management servers, and management clients must be accessible from one another using IPv4. In addition, SQL nodes must use IPv4 to communicate with the cluster. In NDB 8.0.22 and later, these restrictions no
NDB Cluster Replication Schema and Tables

Attribute promotion and demotion. NDB Cluster Replication includes support for attribute promotion and demotion. The implementation of the latter distinguishes between lossy and non-lossy type conversions, and their use on the replica can be controlled by setting the `slave_type_conversions` global server system variable.

For more information about attribute promotion and demotion in NDB Cluster, see Row-based replication: attribute promotion and demotion.

NDB, unlike InnoDB or MyISAM, does not write changes to virtual columns to the binary log; however, this has no detrimental effects on NDB Cluster Replication or replication between NDB and other storage engines. Changes to stored generated columns are logged.

8.4 NDB Cluster Replication Schema and Tables

Replication in NDB Cluster makes use of a number of dedicated tables in the `mysql` database on each MySQL Server instance acting as an SQL node in both the cluster being replicated and in the replica. This is true regardless of whether the replica is a single server or a cluster. These tables are created during the MySQL installation process, and include a table for storing the binary log's indexing data. Since the `ndb_binlog_index` table is local to each MySQL server and does not participate in clustering, it uses the InnoDB storage engine. This means that it must be created separately on each `mysqld` participating in the source cluster. (The binary log itself contains updates from all MySQL servers in the cluster to be replicated.) This table is defined as follows:

```sql
CREATE TABLE `ndb_binlog_index` (
  `Position` BIGINT(20) UNSIGNED NOT NULL,
  `File` VARCHAR(255) NOT NULL,
  `epoch` BIGINT(20) UNSIGNED NOT NULL,
  `inserts` INT(10) UNSIGNED NOT NULL,
  `updates` INT(10) UNSIGNED NOT NULL,
  `deletes` INT(10) UNSIGNED NOT NULL,
  `schemaops` INT(10) UNSIGNED NOT NULL,
  `orig_server_id` INT(10) UNSIGNED NOT NULL,
  `orig_epoch` BIGINT(20) UNSIGNED NOT NULL,
  `gci` INT(10) UNSIGNED NOT NULL,
  `next_position` bigint(20) unsigned NOT NULL,
  `next_file` varchar(255) NOT NULL,
) ENGINE=InnoDB DEFAULT CHARSET=latin1;
```

Note

If you are upgrading from an older release (prior to NDB 7.5.2), perform the MySQL upgrade procedure and ensure that the system tables are upgraded. (As of MySQL 8.0.16, start the server with the `--upgrade=FORCE` option. Prior to MySQL 8.0.16, invoke `mysql_upgrade` with the `--force` and `--upgrade-system-tables` options after starting the server.) The system table upgrade causes an `ALTER TABLE ... ENGINE=INNODB` statement to be executed for this table. Use of the MyISAM storage engine for this table continues to be supported for backward compatibility.

`ndb_binlog_index` may require additional disk space after being converted to InnoDB. If this becomes an issue, you may be able to conserve space by using an InnoDB tablespace for this table, changing its `ROW_FORMAT` to `COMPRESSED`, or both. For more information, see `CREATE TABLESPACE Statement`, and `CREATE TABLE Statement`, as well as Tablespaces.

The size of the `ndb_binlog_index` table is dependent on the number of epochs per binary log file and the number of binary log files. The number of epochs per binary log file normally depends on the amount of binary log generated per epoch and the size of the binary log file, with smaller epochs
resulting in more epochs per file. You should be aware that empty epochs produce inserts to the
ndb_binlog_index table, even when the --ndb-log-empty-epochs option is OFF, meaning that the
number of entries per file depends on the length of time that the file is in use; this relationship can
be represented by the formula shown here:

\[ \text{number of epochs per file} = \frac{\text{time spent per file}}{\text{TimeBetweenEpochs}} \]

A busy NDB Cluster writes to the binary log regularly and presumably rotates binary log files more
quickly than a quiet one. This means that a “quiet” NDB Cluster with --ndb-log-empty-epochs=ON
can actually have a much higher number of ndb_binlog_index rows per file than one with a great
deal of activity.

When mysqld is started with the --ndb-log-orig option, the orig_server_id and orig_epoch
columns store, respectively, the ID of the server on which the event originated and the epoch in
which the event took place on the originating server, which is useful in NDB Cluster replication setups
employing multiple sources. The SELECT statement used to find the closest binary log position to
the highest applied epoch on the replica in a multi-source setup (see Section 8.10, “NDB Cluster
Replication: Bidirectional and Circular Replication”) employs these two columns, which are not indexed.
This can lead to performance issues when trying to fail over, since the query must perform a table
scan, especially when the source has been running with --ndb-log-empty-epochs=ON. You can
improve multi-source failover times by adding an index to these columns, as shown here:

```
ALTER TABLE mysql.ndb_binlog_index
ADD INDEX orig_lookup USING BTREE (orig_server_id, orig_epoch);
```

Adding this index provides no benefit when replicating from a single source to a single replica, since
the query used to get the binary log position in such cases makes no use of orig_server_id or
orig_epoch.

See Section 8.8, “Implementing Failover with NDB Cluster Replication”, for more information about
using the next_position and next_file columns.

The following figure shows the relationship of the NDB Cluster replication source server, its binary log
injector thread, and the mysql.ndb_binlog_index table.

Figure 8.5 The Replication Source Cluster
Preparing the NDB Cluster for Replication

An additional table, named `ndb_apply_status`, is used to keep a record of the operations that have been replicated from the source to the replica. Unlike the case with `ndb_binlog_index`, the data in this table is not specific to any one SQL node in the (replica) cluster, and so `ndb_apply_status` can use the NDBCLUSTER storage engine, as shown here:

```sql
CREATE TABLE `ndb_apply_status` (
    `server_id`   INT(10) UNSIGNED NOT NULL,
    `epoch`       BIGINT(20) UNSIGNED NOT NULL,
    `log_name`    VARCHAR(255) CHARACTER SET latin1 COLLATE latin1_bin NOT NULL,
    `start_pos`   BIGINT(20) UNSIGNED NOT NULL,
    `end_pos`     BIGINT(20) UNSIGNED NOT NULL,
    PRIMARY KEY (`server_id`) USING HASH
) ENGINE=NDBCLUSTER DEFAULT CHARSET=latin1;
```

The `ndb_apply_status` table is populated only on replicas, which means that, on the source, this table never contains any rows; thus, there is no need to allot any DataMemory to `ndb_apply_status` there.

Because this table is populated from data originating on the source, it should be allowed to replicate; any replication filtering or binary log filtering rules that inadvertently prevent the replica from updating `ndb_apply_status`, or that prevent the source from writing into the binary log may prevent replication between clusters from operating properly. For more information about potential problems arising from such filtering rules, see Replication and binary log filtering rules with replication between NDB Clusters.

The `ndb_binlog_index` and `ndb_apply_status` tables are created in the mysql database because they should not be explicitly replicated by the user. User intervention is normally not required to create or maintain either of these tables, since both are maintained by the NDB binary log (binlog) injector thread. This keeps the source mysql process updated to changes performed by the NDB storage engine. The NDB binlog injector thread receives events directly from the NDB storage engine. The NDB injector is responsible for capturing all the data events within the cluster, and ensures that all events which change, insert, or delete data are recorded in the `ndb_binlog_index` table. The replica I/O thread transfers the events from the source’s binary log to the replica’s relay log.

Even though `ndb_binlog_index` and `ndb_apply_status` are created and maintained automatically, it is advisable to check for the existence and integrity of these tables as an initial step in preparing an NDB Cluster for replication. It is possible to view event data recorded in the binary log by querying the `mysql.ndb_binlog_index` table directly on the source. This can be also be accomplished using the `SHOW BINLOG EVENTS` statement on either the source or replica SQL node. (See SHOW BINLOG EVENTS Statement.)

You can also obtain useful information from the output of `SHOW ENGINE NDB STATUS`.

**Note**
When performing schema changes on NDB tables, applications should wait until the `ALTER TABLE` statement has returned in the MySQL client connection that issued the statement before attempting to use the updated definition of the table.

If the `ndb_apply_status` table does not exist on the replica, `ndb_restore` re-creates it.

Conflict resolution for NDB Cluster Replication requires the presence of an additional `mysql.ndb_replication` table. Currently, this table must be created manually. For information about how to do this, see Section 8.11, “NDB Cluster Replication Conflict Resolution”.

### 8.5 Preparing the NDB Cluster for Replication

Preparing the NDB Cluster for replication consists of the following steps:

1. Check all MySQL servers for version compatibility (see Section 8.2, “General Requirements for NDB Cluster Replication”).
2. Create a replication account on the source Cluster with the appropriate privileges, using the following two SQL statements:

```sql
mysql> CREATE USER 'replica_user'@'replica_host'
    -> IDENTIFIED BY 'replica_password';
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'replica_user'@'replica_host';
```

In the previous statement, `replica_user` is the replication account user name, `replica_host` is the host name or IP address of the replica, and `replica_password` is the password to assign to this account.

For example, to create a replica user account with the name `myreplica`, logging in from the host named `replica-host`, and using the password `53cr37`, use the following CREATE USER and GRANT statements:

```sql
mysql> CREATE USER 'myreplica'@'replica-host'
    -> IDENTIFIED BY '53cr37';
mysql> GRANT REPLICATION SLAVE ON *.*
    -> TO 'myreplica'@'replica-host';
```

For security reasons, it is preferable to use a unique user account—not employed for any other purpose—for the replication account.

3. Set up the replica to use the source. Using the mysql client, this can be accomplished with the `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23):

```sql
mysql> CHANGE MASTER TO
    -> MASTER_HOST='source_host',
    -> MASTER_PORT=source_port,
    -> MASTER_USER='replica_user',
    -> MASTER_PASSWORD='replica_password';
```

Or from MySQL 8.0.23:

```sql
mysql> CHANGE REPLICATION SOURCE TO
    -> SOURCE_HOST='source_host',
    -> SOURCE_PORT=source_port,
    -> SOURCE_USER='replica_user',
    -> SOURCE_PASSWORD='replica_password';
```

In the previous statement, `source_host` is the host name or IP address of the replication source, `source_port` is the port for the replica to use when connecting to the source, `replica_user` is the user name set up for the replica on the source, and `replica_password` is the password set for that user account in the previous step.

For example, to tell the replica to use the MySQL server whose host name is `rep-source` with the replication account created in the previous step, use the following statement:

```sql
mysql> CHANGE MASTER TO
    -> MASTER_HOST='rep-source',
    -> MASTER_PORT=3306,
    -> MASTER_USER='myreplica',
    -> MASTER_PASSWORD='53cr37';
```

Or from MySQL 8.0.23:

```sql
mysql> CHANGE REPLICATION SOURCE TO
    -> SOURCE_HOST='rep-source',
    -> SOURCE_PORT=3306,
    -> SOURCE_USER='myreplica',
```
Starting NDB Cluster Replication (Single Replication Channel)

For a complete list of options that can be used with this statement, see CHANGE MASTER TO Statement.

To provide replication backup capability, you also need to add an --ndb-connectstring option to the replica's my.cnf file prior to starting the replication process. See Section 8.9, “NDB Cluster Backups With NDB Cluster Replication”, for details.

For additional options that can be set in my.cnf for replicas, see Replication and Binary Logging Options and Variables.

4. If the source cluster is already in use, you can create a backup of the source and load this onto the replica to cut down on the amount of time required for the replica to synchronize itself with the source. If the replica is also running NDB Cluster, this can be accomplished using the backup and restore procedure described in Section 8.9, “NDB Cluster Backups With NDB Cluster Replication”.

ndb-connectstring=management_host[:port]

In the event that you are not using NDB Cluster on the replica, you can create a backup with this command on the source:

shell$> mysqldump --master-data=1

Then import the resulting data dump onto the replica by copying the dump file over to it. After this, you can use the mysql client to import the data from the dumpfile into the replica database as shown here, where dump_file is the name of the file that was generated using mysqldump on the source, and db_name is the name of the database to be replicated:

shell$> mysql -u root -p db_name < dump_file

For a complete list of options to use with mysqldump, see mysqldump — A Database Backup Program.

Note

If you copy the data to the replica in this fashion, make sure that you stop the replica from trying to connect to the source to begin replicating before all the data has been loaded. You can do this by starting the replica with the --skip-slave-start option on the command line, by including skip-slave-start in the replica's my.cnf file, or from MySQL 8.0.24, by setting the skip_slave_start system variable. Once the data loading has completed, follow the additional steps outlined in the next two sections.

5. Ensure that each MySQL server acting as a replication source is assigned a unique server ID, and has binary logging enabled, using the row-based format. (See Replication Formats.) In addition, we recommend enabling the slave_allow_batching system variable, and possibly increasing the values used with the --ndb-batch-size and --ndb-blob-write-batch-bytes options as well. All of these options can be set either in the source server's my.cnf file, or on the command line when starting the source mysqld process. See Section 8.6, “Starting NDB Cluster Replication (Single Replication Channel)”, for more information.

8.6 Starting NDB Cluster Replication (Single Replication Channel)

This section outlines the procedure for starting NDB Cluster replication using a single replication channel.

1. Start the MySQL replication source server by issuing this command:

shell$> mysqld --ndbcluster --server-id=id --log-bin --ndb-log-bin &
In the previous statement, \textit{id} is this server's unique ID (see Section 8.2, "General Requirements for NDB Cluster Replication"). This starts the server's \texttt{mysqld} process with binary logging enabled using the proper logging format. It is also necessary in NDB 8.0 to enable logging of updates to NDB tables explicitly, using the \texttt{--ndb-log-bin} option; this is a change from previous versions of NDB Cluster, in which this option was enabled by default.

\textbf{Note}

You can also start the source with \texttt{--binlog-format=MIXED}, in which case row-based replication is used automatically when replicating between clusters. Statement-based binary logging is not supported for NDB Cluster Replication (see Section 8.2, "General Requirements for NDB Cluster Replication").

2. Start the MySQL replica server as shown here:

```
shell> mysqld --ndbcluster --server-id=id &
```

In the command just shown, \textit{id} is the replica server's unique ID. It is not necessary to enable logging on the replica.

\textbf{Note}

Unless you want replication to begin immediately, delay the start of the replication threads until the appropriate \texttt{START REPLICA | SLAVE} statement has been issued, as explained in Step 4 below. You can do this by starting the replica with the \texttt{--skip-slave-start} option on the command line, by including \texttt{skip-slave-start} in the replica's \texttt{my.cnf} file, or from MySQL 8.0.24, by setting the \texttt{skip_slave_start} system variable.

3. It is necessary to synchronize the replica server with the source server's replication binary log. If binary logging has not previously been running on the source, run the following statement on the replica:

```
mysql> CHANGE MASTER TO
    -> MASTER_LOG_FILE=''
    -> MASTER_LOG_POS=4;
Or from MySQL 8.0.23:
mysql> CHANGE REPLICATION SOURCE TO
    -> SOURCE_LOG_FILE=''
    -> SOURCE_LOG_POS=4;
```

This instructs the replica to begin reading the source server's binary log from the log's starting point. Otherwise—that is, if you are loading data from the source using a backup—see Section 8.8, "Implementing Failover with NDB Cluster Replication", for information on how to obtain the correct values to use for \texttt{MASTER_LOG_FILE} and \texttt{MASTER_LOG_POS} in such cases.

4. Finally, instruct the replica to begin applying replication by issuing this command from the \texttt{mysql} client on the replica:

```
mysql> START SLAVE;
Or from MySQL 8.0.22:
mysql> START REPLICA;
```

This also initiates the transmission of data and changes from the source to the replica.

It is also possible to use two replication channels, in a manner similar to the procedure described in the next section; the differences between this and using a single replication channel are covered in Section 8.7, "Using Two Replication Channels for NDB Cluster Replication".
It is also possible to improve cluster replication performance by enabling *batched updates*. This can be accomplished by setting the `slave_allow_batching` system variable on the replicas' `mysqld` processes. Normally, updates are applied as soon as they are received. However, the use of batching causes updates to be applied in batches of 32 KB each; this can result in higher throughput and less CPU usage, particularly where individual updates are relatively small.

**Note**

Batching works on a per-epoch basis; updates belonging to more than one transaction can be sent as part of the same batch. All outstanding updates are applied when the end of an epoch is reached, even if the updates total less than 32 KB.

Batching can be turned on and off at runtime. To activate it at runtime, you can use either of these two statements:

```sql
SET GLOBAL slave_allow_batching = 1;
SET GLOBAL slave_allow_batching = ON;
```

If a particular batch causes problems (such as a statement whose effects do not appear to be replicated correctly), batching can be deactivated using either of the following statements:

```sql
SET GLOBAL slave_allow_batching = 0;
SET GLOBAL slave_allow_batching = OFF;
```

You can check whether batching is currently being used by means of an appropriate `SHOW VARIABLES` statement, like this one:

```sql
mysql> SHOW VARIABLES LIKE 'slave%';
+---------------------------+-------+
| Variable_name             | Value |
|---------------------------+-------+
| slave_allow_batching      | ON    |
| slave_compressed_protocol | OFF   |
| slave_load_tmpdir         | /tmp  |
| slave_net_timeout         | 3600  |
| slave_skip_errors         | OFF   |
| slave_transaction_retries | 10    |
+---------------------------+-------+
6 rows in set (0.00 sec)
```

8.7 Using Two Replication Channels for NDB Cluster Replication

In a more complete example scenario, we envision two replication channels to provide redundancy and thereby guard against possible failure of a single replication channel. This requires a total of four replication servers, two source servers on the source cluster and two replica servers on the replica cluster. For purposes of the discussion that follows, we assume that unique identifiers are assigned as shown here:

<table>
<thead>
<tr>
<th>Server ID</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>Source - primary replication channel (<em>S</em>)</td>
</tr>
<tr>
<td>2</td>
<td>Source - secondary replication channel (*S')</td>
</tr>
<tr>
<td>3</td>
<td>Replica - primary replication channel (<em>R</em>)</td>
</tr>
<tr>
<td>4</td>
<td>replica - secondary replication channel (*R')</td>
</tr>
</tbody>
</table>

Setting up replication with two channels is not radically different from setting up a single replication channel. First, the `mysqld` processes for the primary and secondary replication source servers must be started, followed by those for the primary and secondary replicas. The replication processes can be
initiated by issuing the `START REPLICA | SLAVE` statement on each of the replicas. The commands and the order in which they need to be issued are shown here:

1. Start the primary replication source:
   ```
   shell
   $ mysqld --ndbcluster --server-id=1 \
       --log-bin &
   ```

2. Start the secondary replication source:
   ```
   shell
   $ mysqld --ndbcluster --server-id=2 \
       --log-bin &
   ```

3. Start the primary replica server:
   ```
   shell
   $ mysqld --ndbcluster --server-id=3 \
       --skip-slave-start &
   ```

4. Start the secondary replica server:
   ```
   shell
   $ mysqld --ndbcluster --server-id=4 \
       --skip-slave-start &
   ```

5. Finally, initiate replication on the primary channel by executing the `START REPLICA | SLAVE` statement on the primary replica as shown here:
   ```
   mysql
   $ START SLAVE;
   Or from MySQL 8.0.22:
   mysql
   $ START REPLICA;
   ```

**Warning**

Only the primary channel must be started at this point. The secondary replication channel needs to be started only in the event that the primary replication channel fails, as described in Section 8.8, “Implementing Failover with NDB Cluster Replication”. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replicas.

As mentioned previously, it is not necessary to enable binary logging on the replicas.

### 8.8 Implementing Failover with NDB Cluster Replication

In the event that the primary Cluster replication process fails, it is possible to switch over to the secondary replication channel. The following procedure describes the steps required to accomplish this.

1. Obtain the time of the most recent global checkpoint (GCP). That is, you need to determine the most recent epoch from the `ndb_apply_status` table on the replica cluster, which can be found using the following query:
   ```
   mysql
   $ SELECT @latest:=MAX(epoch) \
       FROM mysql.ndb_apply_status;
   ```

In a circular replication topology, with a source and a replica running on each host, when you are using `ndb_log_apply_status=1`, NDB Cluster epochs are written in the replicas’ binary logs. This means that the `ndb_apply_status` table contains information for the replica on this host as well as for any other host which acts as a replica of the replication source server running on this host.

In this case, you need to determine the latest epoch on this replica to the exclusion of any epochs from any other replicas in this replica's binary log that were not listed in the `IGNORE_SERVER_IDS` options of the `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` statement used to set up this replica. The reason for excluding such epochs is that rows in the
mysql.ndb_apply_status table whose server IDs have a match in the IGNORE_SERVER_IDS list from the CHANGE REPLICA SOURCE TO | CHANGE MASTER TO statement used to prepare this replica's source are also considered to be from local servers, in addition to those having the replica's own server ID. You can retrieve this list as Replicate_Ignore_Server_Ids from the output of SHOW REPLICA | SLAVE STATUS. We assume that you have obtained this list and are substituting it for ignore_server_ids in the query shown here, which like the previous version of the query, selects the greatest epoch into a variable named @latest:

mysqlR> SELECT @latest:=MAX(epoch)
  -> FROM mysql.ndb_apply_status
  -> WHERE server_id NOT IN (ignore_server_ids);

In some cases, it may be simpler or more efficient (or both) to use a list of the server IDs to be included and server_id IN server_id_list in the WHERE condition of the preceding query.

2. Using the information obtained from the query shown in Step 1, obtain the corresponding records from the ndb_binlog_index table on the source cluster.

You can use the following query to obtain the needed records from the ndb_binlog_index table on the source:

mysqlS> SELECT
  -> @file:=SUBSTRING_INDEX(next_file, '/', -1),
  -> @pos:=next_position
  -> FROM mysql.ndb_binlog_index
  -> WHERE epoch >= @latest
  -> ORDER BY epoch ASC LIMIT 1;

These are the records saved on the source since the failure of the primary replication channel. We have employed a user variable @latest here to represent the value obtained in Step 1. Of course, it is not possible for one mysqld instance to access user variables set on another server instance directly. These values must be “plugged in” to the second query manually or by an application.

Important
You must ensure that the replica mysqld is started with --slave-skip-errors=ddl_exist_errors before executing START REPLICA | SLAVE. Otherwise, replication may stop with duplicate DDL errors.

3. Now it is possible to synchronize the secondary channel by running the following query on the secondary replica server:

mysqlR> CHANGE MASTER TO
  -> MASTER_LOG_FILE='@file',
  -> MASTER_LOG_POS=@pos;
Or from MySQL 8.0.23:
mysqlR> CHANGE REPLICA SOURCE TO
  -> SOURCE_LOG_FILE='@file',
  -> SOURCE_LOG_POS=@pos;

Again we have employed user variables (in this case @file and @pos) to represent the values obtained in Step 2 and applied in Step 3; in practice these values must be inserted manually or using an application that can access both of the servers involved.

Important
You must ensure that the replica mysqld is started with --slave-skip-errors=ddl_exist_errors before executing START REPLICA | SLAVE. Otherwise, replication may stop with duplicate DDL errors.

4. You can now initiate replication on the secondary channel by issuing the appropriate command on the secondary replica mysqld:
Once the secondary replication channel is active, you can investigate the failure of the primary and effect repairs. The precise actions required to do this depend upon the reasons for which the primary channel failed.

**Warning**

The secondary replication channel is to be started only if and when the primary replication channel has failed. Running multiple replication channels simultaneously can result in unwanted duplicate records being created on the replicas.

If the failure is limited to a single server, it should in theory be possible to replicate from \( S \) to \( R' \), or from \( S' \) to \( R \).

8.9 NDB Cluster Backups With NDB Cluster Replication

This section discusses making backups and restoring from them using NDB Cluster replication. We assume that the replication servers have already been configured as covered previously (see Section 8.5, “Preparing the NDB Cluster for Replication”, and the sections immediately following). This having been done, the procedure for making a backup and then restoring from it is as follows:

1. There are two different methods by which the backup may be started.

   - **Method A.** This method requires that the cluster backup process was previously enabled on the source server, prior to starting the replication process. This can be done by including the following line in a `[mysql_cluster]` section in the `my.cnf` file, where `management_host` is the IP address or host name of the NDB management server for the source cluster, and `port` is the management server's port number:

     ```
     ndb-connectstring=management_host[:port]
     ```

   - **Method B.** If the `my.cnf` file does not specify where to find the management host, you can start the backup process by passing this information to the NDB management client as part of the `START BACKUP` command. This can be done as shown here, where `management_host` and `port` are the host name and port number of the management server:

     ```
     ndb_mgm management_host:port -e "START BACKUP"
     ```

     In this case, the backup can be started by executing this statement on the replication source:

     ```
     ndb_mgm -e "START BACKUP"
     ```

2. Copy the cluster backup files to the replica that is being brought on line. Each system running an `ndbd` process for the source cluster has cluster backup files located on it, and all of these files must be copied to the replica to ensure a successful restore. The backup files can be copied into any directory on the computer where the replica's management host resides, as long as the MySQL and
NDB binaries have read permissions in that directory. In this case, we assume that these files have been copied into the directory `/var/BACKUPS/BACKUP-1`.

While it is not necessary that the replica cluster have the same number of `ndbd` processes (data nodes) as the source, it is highly recommended this number be the same. It is necessary that the replication process is prevented from starting when the replica server starts. You can do this by starting the replica with the `--skip-slave-start` option on the command line, by including `skip-slave-start` in the replica's `my.cnf` file, or from MySQL 8.0.24, by setting the `skip_slave_start` system variable.

3. Create any databases on the replica cluster that are present on the source cluster and that are to be replicated.

   **Important**
   
   A `CREATE DATABASE` (or `CREATE SCHEMA`) statement corresponding to each database to be replicated must be executed on each SQL node in the replica cluster.

4. Reset the replica cluster using this statement in the `mysql` client:

   ```
   mysql> RESET SLAVE;
   Or from MySQL 8.0.22:
   mysql> RESET REPLICA;
   ```

5. You can now start the cluster restoration process on the replica using the `ndb_restore` command for each backup file in turn. For the first of these, it is necessary to include the `-m` option to restore the cluster metadata, as shown here:

   ```
   shell> ndb_restore -c replica_host:port -n node-id \ 
         -b backup-id -m -r dir
   ```

   `dir` is the path to the directory where the backup files have been placed on the replica. For the `ndb_restore` commands corresponding to the remaining backup files, the `-m` option should **not** be used.

   For restoring from a source cluster with four data nodes (as shown in the figure in Chapter 8, *NDB Cluster Replication*) where the backup files have been copied to the directory `/var/BACKUPS/BACKUP-1`, the proper sequence of commands to be executed on the replica might look like this:

   ```
   shell> ndb_restore -c replica-host:1186 -n 2 -b 1 -m \ 
          -r ./var/BACKUPS/BACKUP-1
   shell> ndb_restore -c replica-host:1186 -n 3 -b 1 \ 
          -r ./var/BACKUPS/BACKUP-1
   shell> ndb_restore -c replica-host:1186 -n 4 -b 1 \ 
          -r ./var/BACKUPS/BACKUP-1
   shell> ndb_restore -c replica-host:1186 -n 5 -b 1 -e \ 
          -r ./var/BACKUPS/BACKUP-1
   ```

   **Important**
   
   The `-e` (or `--restore-epoch`) option in the final invocation of `ndb_restore` in this example is required to make sure that the epoch is written to the replica's `mysql.ndb_apply_status` table. Without this information, the replica cannot synchronize properly with the source. (See Section 6.23, "ndb_restore — Restore an NDB Cluster Backup").

6. Now you need to obtain the most recent epoch from the `ndb_apply_status` table on the replica (as discussed in Section 8.8, "Implementing Failover with NDB Cluster Replication"):

   ```
   mysql> SELECT @latest:=MAX(epoch) 
       FROM mysql.ndb_apply_status;
   ```
7. Using `@latest` as the epoch value obtained in the previous step, you can obtain the correct starting position `@pos` in the correct binary log file `@file` from the `mysql.ndb_binlog_index` table on the source using the query shown here:

```
mysql> SELECT
    -> @file:=SUBSTRING_INDEX(File, '/', -1),
    -> @pos:=Position
    -> FROM mysql.ndb_binlog_index
    -> WHERE epoch >= @latest
    -> ORDER BY epoch ASC LIMIT 1;
```

In the event that there is currently no replication traffic, you can get this information by running `SHOW MASTER STATUS` on the source and using the value shown in the `Position` column of the output for the file whose name has the suffix with the greatest value for all files shown in the `File` column. In this case, you must determine which file this is and supply the name in the next step manually or by parsing the output with a script.

8. Using the values obtained in the previous step, you can now issue the appropriate `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23) in the replica's `mysql` client:

```
mysql> CHANGE MASTER TO
    -> MASTER_LOG_FILE='@file',
    -> MASTER_LOG_POS=@pos;
```

Or from MySQL 8.0.23:
```
mysql> CHANGE REPLICATION SOURCE TO
    -> SOURCE_LOG_FILE='@file',
    -> SOURCE_LOG_POS=@pos;
```

9. Now that the replica knows from what point in which binary log file to start reading data from the source, you can cause the replica to begin replicating with this statement:

```
mysql> START SLAVE;
```

Or from MySQL 8.0.22:
```
mysql> START REPLICA;
```

To perform a backup and restore on a second replication channel, it is necessary only to repeat these steps, substituting the host names and IDs of the secondary source and replica for those of the primary source and replica servers where appropriate, and running the preceding statements on them.

For additional information on performing Cluster backups and restoring Cluster from backups, see Section 7.8, “Online Backup of NDB Cluster”.

### 8.9.1 NDB Cluster Replication: Automating Synchronization of the Replica to the Source Binary Log

It is possible to automate much of the process described in the previous section (see Section 8.9, “NDB Cluster Backups With NDB Cluster Replication”). The following Perl script `reset-replica.pl` serves as an example of how you can do this.

```perl
#!/usr/bin/perl -w
# file: reset-replica.pl
#
# Copyright (c) 2005, 2020, Oracle and/or its affiliates. All rights reserved.
# This program is free software; you can redistribute it and/or modify
# it under the terms of the GNU General Public License as published by
# the Free Software Foundation; either version 2 of the License, or
# (at your option) any later version.
# This program is distributed in the hope that it will be useful,
# but WITHOUT ANY WARRANTY; without even the implied warranty of
# MERCHANTABILITY or FITNESS FOR A PARTICULAR PURPOSE. See the
# GNU General Public License for more details.
# You should have received a copy of the GNU General Public License
# along with this program; if not, write to:
# Free Software Foundation, Inc.
# 59 Temple Place, Suite 330
# Boston, MA 02111-1307 USA
```
# Version 1.1

#### Includes
```
use DBI;
```

#### Globals
```
my $m_host='';
my $m_port='';
my $m_user='';
my $m_pass='';
my $s_host='';
my $s_port='';
my $s_user='';
my $s_pass='';
my $dbhM='';
my $dbhS='';
```

#### Sub Prototypes
```
sub CollectCommandPromptInfo;
sub ConnectToDatabases;
sub DisconnectFromDatabases;
sub GetReplicaEpoch;
sub GetSourceInfo;
sub UpdateReplica;
```

#### Program Main
```
CollectCommandPromptInfo;
ConnectToDatabases;
GetReplicaEpoch;
GetSourceInfo;
UpdateReplica;
DisconnectFromDatabases;
```

#### Collect Command Prompt Info
```
sub CollectCommandPromptInfo
{
### Check that user has supplied correct number of command line args
    die "Usage:
    reset-replica >source MySQL host< >source MySQL port<
    >source user< >source pass< >replica MySQL host<
    >replica MySQL port< >replica user< >replica pass<
    All 8 arguments must be passed. Use BLANK for NULL passwords"
    unless @ARGV == 8;
    $m_host  =  $ARGV[0];
    $m_port  =  $ARGV[1];
    $m_user  =  $ARGV[2];
    $m_pass  =  $ARGV[3];
    $s_host  =  $ARGV[4];
    $s_port  =  $ARGV[5];
    $s_user  =  $ARGV[6];
    $s_pass  =  $ARGV[7];
    if ($m_pass eq "BLANK") { $m_pass = '';}
    if ($s_pass eq "BLANK") { $s_pass = '';}
}
```

#### Make connections to both databases
```
sub ConnectToDatabases
{
### Connect to both source and replica cluster databases
### Connect to source
    $dbhM = DBI->connect(
    "dbi:mysql:database=mysql;host=$m_host;port=$m_port",
    "$m_user", "$m_pass")
    or die "Can't connect to source cluster MySQL process!
    Error: $DBI::errstr"
;
### Connect to replica
    $dbhS = DBI->connect(
    "dbi:mysql:database=mysql;host=$s_host",
    "$s_user", "$s_pass")
    or die "Can't connect to replica cluster MySQL process!
    Error: $DBI::errstr"
;
}
```

#### Disconnect from both databases
```
sub DisconnectFromDatabases
{
### Disconnect from source
```
Point-In-Time Recovery Using NDB Cluster Replication

Point-in-time recovery—that is, recovery of data changes made since a given point in time—is performed after restoring a full backup that returns the server to its state when the backup was made. Performing point-in-time recovery of NDB Cluster tables with NDB Cluster and NDB Cluster Replication can be accomplished using a native NDB data backup (taken by issuing `CREATE BACKUP` in the `ndb_mgm` client) and restoring the `ndb_binlog_index` table (from a dump made using `mysqldump`).

To perform point-in-time recovery of NDB Cluster, it is necessary to follow the steps shown here:

1. Back up all NDB databases in the cluster, using the `START_BACKUP` command in the `ndb_mgm` client (see Section 7.8, “Online Backup of NDB Cluster”).
2. At some later point, prior to restoring the cluster, make a backup of the `mysql.ndb_binlog_index` table. It is probably simplest to use `mysqldump` for this task. Also back up the binary log files at this time.
This backup should be updated regularly—perhaps even hourly—depending on your needs.

3. *(Catastrophic failure or error occurs.)*

4. Locate the last known good backup.

5. Clear the data node file systems (using `ndbd --initial` or `ndbmtd --initial`).

   **Note**
   Beginning with NDB 8.0.21, Disk Data tablespace and log files are removed by `--initial`. Previously, it was necessary to delete these manually.

6. Use `DROP TABLE` or `TRUNCATE TABLE` with the `mysql.ndb_binlog_index` table.

7. Execute `ndb_restore`, restoring all data. You must include the `--restore-epoch` option when you run `ndb_restore`, so that the `ndb_apply_status` table is populated correctly. (See Section 6.23, “`ndb_restore` — Restore an NDB Cluster Backup”, for more information.)

8. Restore the `ndb_binlog_index` table from the output of `mysqldump` and restore the binary log files from backup, if necessary.

9. Find the epoch applied most recently—that is, the maximum `epoch` column value in the `ndb_apply_status` table—as the user variable `@LATEST_EPOCH` (emphasized):

   ```sql
   SELECT @LATEST_EPOCH:=MAX(epoch)
   FROM mysql.ndb_apply_status;
   ```

10. Find the latest binary log file (`@FIRST_FILE`) and position (`Position` column value) within this file that correspond to `@LATEST_EPOCH` in the `ndb_binlog_index` table:

   ```sql
   SELECT Position, @FIRST_FILE:=File
   FROM mysql.ndb_binlog_index
   WHERE epoch > @LATEST_EPOCH ORDER BY epoch ASC LIMIT 1;
   ```

11. Using `mysqlbinlog`, replay the binary log events from the given file and position up to the point of the failure. (See `mysqlbinlog` — Utility for Processing Binary Log Files.)

   See also Point-in-Time (Incremental) Recovery, for more information about the binary log, replication, and incremental recovery.

### 8.10 NDB Cluster Replication: Bidirectional and Circular Replication

It is possible to use NDB Cluster for bidirectional replication between two clusters, as well as for circular replication between any number of clusters.

#### Circular replication example.

In the next few paragraphs we consider the example of a replication setup involving three NDB Clusters numbered 1, 2, and 3, in which Cluster 1 acts as the replication source for Cluster 2, Cluster 2 acts as the source for Cluster 3, and Cluster 3 acts as the source for Cluster 1. Each cluster has two SQL nodes, with SQL nodes A and B belonging to Cluster 1, SQL nodes C and D belonging to Cluster 2, and SQL nodes E and F belonging to Cluster 3.

Circular replication using these clusters is supported as long as the following conditions are met:

- The SQL nodes on all sources and replicas are the same.
- All SQL nodes acting as sources and replicas are started with the `log_slave_updates` system variable enabled.

This type of circular replication setup is shown in the following diagram:
In this scenario, SQL node A in Cluster 1 replicates to SQL node C in Cluster 2; SQL node C replicates to SQL node E in Cluster 3; SQL node E replicates to SQL node A. In other words, the replication line (indicated by the curved arrows in the diagram) directly connects all SQL nodes used as replication sources and replicas.

It is also possible to set up circular replication in such a way that not all source SQL nodes are also replicas, as shown here:
In this case, different SQL nodes in each cluster are used as replication sources and replicas. You must *not* start any of the SQL nodes with the `log_slave_updates` system variable enabled. This type of circular replication scheme for NDB Cluster, in which the line of replication (again indicated by the curved arrows in the diagram) is discontinuous, should be possible, but it should be noted that it has not yet been thoroughly tested and must therefore still be considered experimental.

**Using NDB-native backup and restore to initialize a replica cluster.** When setting up circular replication, it is possible to initialize the replica cluster by using the management client `START BACKUP` command on one NDB Cluster to create a backup and then applying this backup on another NDB Cluster using `ndb_restore`. This does not automatically create binary logs on the second NDB Cluster's SQL node acting as the replica; in order to cause the binary logs to be created, you must issue a `SHOW TABLES` statement on that SQL node; this should be done prior to running `START REPLICA | SLAVE`. This is a known issue.

**Multi-source failover example.** In this section, we discuss failover in a multi-source NDB Cluster replication setup with three NDB Clusters having server IDs 1, 2, and 3. In this scenario, Cluster 1 replicates to Clusters 2 and 3; Cluster 2 also replicates to Cluster 3. This relationship is shown here:
In other words, data replicates from Cluster 1 to Cluster 3 through 2 different routes: directly, and by way of Cluster 2.

Not all MySQL servers taking part in multi-source replication must act as both source and replica, and a given NDB Cluster might use different SQL nodes for different replication channels. Such a case is shown here:

MySQL servers acting as replicas must be run with the `log_slave_updates` system variable enabled. Which `mysqld` processes require this option is also shown in the preceding diagram.

**Note**

Using the `log_slave_updates` system variable has no effect on servers not being run as replicas.

The need for failover arises when one of the replicating clusters goes down. In this example, we consider the case where Cluster 1 is lost to service, and so Cluster 3 loses 2 sources of updates from Cluster 1. Because replication between NDB Clusters is asynchronous, there is no guarantee that Cluster 3’s updates originating directly from Cluster 1 are more recent than those received through Cluster 2. You can handle this by ensuring that Cluster 3 catches up to Cluster 2 with regard to updates from Cluster 1. In terms of MySQL servers, this means that you need to replicate any outstanding updates from MySQL server C to server F.
On server C, perform the following queries:

```sql
mysqlC> SELECT @latest:=MAX(epoch)
    ->     FROM mysql.ndb_apply_status
    ->     WHERE server_id=1;
mysqlC> SELECT
    ->     @file:=SUBSTRING_INDEX(File, '/', -1),
    ->     @pos:=Position
    ->     FROM mysql.ndb_binlog_index
    ->     WHERE orig_epoch >= @latest
    ->     AND orig_server_id = 1
    ->     ORDER BY epoch ASC LIMIT 1;
```

Note

You can improve the performance of this query, and thus likely speed up failover times significantly, by adding the appropriate index to the `ndb_binlog_index` table. See Section 8.4, “NDB Cluster Replication Schema and Tables”, for more information.

Copy over the values for `@file` and `@pos` manually from server C to server F (or have your application perform the equivalent). Then, on server F, execute the following `CHANGE REPLICATION SOURCE TO` statement (from MySQL 8.0.23) or `CHANGE MASTER TO` statement (before MySQL 8.0.23):

```sql
mysqlF> CHANGE MASTER TO
    ->     MASTER_HOST = 'serverC'
    ->     MASTER_LOG_FILE='@file',
    ->     MASTER_LOG_POS=@pos;
```

Or from MySQL 8.0.23:

```sql
mysqlF> CHANGE REPLICATION SOURCE TO
    ->     SOURCE_HOST = 'serverC'
    ->     SOURCE_LOG_FILE='@file',
    ->     SOURCE_LOG_POS=@pos;
```

Once this has been done, you can issue a `START REPLICA | SLAVE` statement on MySQL server F; this causes any missing updates originating from server B to be replicated to server F.

The `CHANGE REPLICATION SOURCE TO | CHANGE MASTER TO` statement also supports an `IGNORE_SERVER_IDS` option which takes a comma-separated list of server IDs and causes events originating from the corresponding servers to be ignored. For more information, see `CHANGE MASTER TO Statement`, and `SHOW SLAVE | REPLICA STATUS Statement`. For information about how this option interacts with the `ndb_log_apply_status` variable, see Section 8.8, “Implementing Failover with NDB Cluster Replication”.

8.11 NDB Cluster Replication Conflict Resolution

When using a replication setup involving multiple sources (including circular replication), it is possible that different sources may try to update the same row on the replica with different data. Conflict resolution in NDB Cluster Replication provides a means of resolving such conflicts by permitting a user-defined resolution column to be used to determine whether or not an update on a given source should be applied on the replica.

Some types of conflict resolution supported by NDB Cluster (`NDB$OLD()`, `NDB$MAX()`, `NDB$MAX_DELETE_WIN()`) implement this user-defined column as a “timestamp” column (although its type cannot be `TIMESTAMP`, as explained later in this section). These types of conflict resolution are always applied a row-by-row basis rather than a transactional basis. The epoch-based conflict resolution functions `NDB$EPOCH()` and `NDB$EPOCH_TRANS()` compare the order in which epochs are replicated (and thus these functions are transactional). Different methods can be used to compare resolution column values on the replica when conflicts occur, as explained later in this section; the method used can be set on a per-table basis.

You should also keep in mind that it is the application’s responsibility to ensure that the resolution column is correctly populated with relevant values, so that the resolution function can make the appropriate choice when determining whether to apply an update.
Logging Full or Partial Rows (--ndb-log-updated-only Option)

Requirements. Preparations for conflict resolution must be made on both the source and the replica. These tasks are described in the following list:

- On the source writing the binary logs, you must determine which columns are sent (all columns or only those that have been updated). This is done for the MySQL Server as a whole by applying the mysqld startup option --ndb-log-updated-only (described later in this section) or on a per-table basis by entries in the mysql.ndb_replication table (see The ndb_replication system table).

  Note
  If you are replicating tables with very large columns (such as TEXT or BLOB columns), --ndb-log-updated-only can also be useful for reducing the size of the binary logs and avoiding possible replication failures due to exceeding max_allowed_packet.

  See Replication and max_allowed_packet, for more information about this issue.

- On the replica, you must determine which type of conflict resolution to apply ("latest timestamp wins", "same timestamp wins", "primary wins", "primary wins, complete transaction", or none). This is done using the mysql.ndb_replication system table, on a per-table basis (see The ndb_replication system table).

- NDB Cluster also supports read conflict detection, that is, detecting conflicts between reads of a given row in one cluster and updates or deletes of the same row in another cluster. This requires exclusive read locks obtained by setting ndb_log_exclusive_reads equal to 1 on the replica. All rows read by a conflicting read are logged in the exceptions table. For more information, see Read conflict detection and resolution.

When using the functions NDB$OLD(), NDB$MAX(), and NDB$MAX_DELETE_WIN() for timestamp-based conflict resolution, we often refer to the column used for determining updates as a “timestamp” column. However, the data type of this column is never TIMESTAMP; instead, its data type should be INT (INTEGER) or BIGINT. The “timestamp” column should also be UNSIGNED and NOT NULL.

The NDB$EPOCH() and NDB$EPOCH_TRANS() functions discussed later in this section work by comparing the relative order of replication epochs applied on a primary and secondary NDB Cluster, and do not make use of timestamps.

Source column control. We can see update operations in terms of “before” and “after” images—that is, the states of the table before and after the update is applied. Normally, when updating a table with a primary key, the “before” image is not of great interest; however, when we need to determine on a per-update basis whether or not to use the updated values on a replica, we need to make sure that both images are written to the source’s binary log. This is done with the --ndb-log-update-as-write option for mysqld, as described later in this section.

Important
Whether logging of complete rows or of updated columns only is done is decided when the MySQL server is started, and cannot be changed online; you must either restart mysql, or start a new mysqld instance with different logging options.

Logging Full or Partial Rows (--ndb-log-updated-only Option)

| Command-Line Format     | --ndb-log-updated-only[={OFF|ON}] |
|-------------------------|-----------------------------------|
| System Variable         | ndb_log_updated_only              |
| Scope                   | Global                            |

564
For purposes of conflict resolution, there are two basic methods of logging rows, as determined by the setting of the `--ndb-log-updated-only` option for `mysqld`:

- Log complete rows
- Log only column data that has been updated—that is, column data whose value has been set, regardless of whether or not this value was actually changed. This is the default behavior.

It is usually sufficient—and more efficient—to log updated columns only; however, if you need to log full rows, you can do so by setting `--ndb-log-updated-only` to 0 or OFF.

The setting of the MySQL Server's `--ndb-log-update-as-write` option determines whether logging is performed with or without the “before” image. Because conflict resolution is done in the MySQL Server's update handler, it is necessary to control logging performed by the replication source such that updates are updates and not writes; that is, such that updates are treated as changes in existing rows rather than the writing of new rows, even though these replace existing rows. This option is turned on by default; in other words, updates are treated as writes. That is, updates are by default written as `write_row` events in the binary log, rather than as `update_row` events.

To disable the option, start the source `mysqld` with `--ndb-log-update-as-write=0` or `--ndb-log-update-as-write=OFF`. You must do this when replicating from NDB tables to tables using a different storage engine; see Replication from NDB to other storage engines, and Replication from NDB to a nontransactional storage engine, for more information.

Conflict resolution control. Conflict resolution is usually enabled on the server where conflicts can occur. Like logging method selection, it is enabled by entries in the `mysql.ndb_replication` table.

The `ndb_replication` system table. To enable conflict resolution, it is necessary to create an `ndb_replication` table in the `mysql` system database on the source, the replica, or both, depending on the conflict resolution type and method to be employed. This table is used to control logging and conflict resolution functions on a per-table basis, and has one row per table involved in replication.

`ndb_replication` is created and filled with control information on the server where the conflict is to be resolved. In a simple source-replica setup where data can also be changed locally on the replica this is typically the replica. In a more complex replication scheme, such as bidirectional replication, this is usually all of the sources involved. Each row in `mysql.ndb_replication` corresponds to a table being replicated, and specifies how to log and resolve conflicts (that is, which conflict resolution function, if any, to use) for that table. The definition of the `mysql.ndb_replication` table is shown here:

```sql
CREATE TABLE mysql.ndb_replication (  
  db VARBINARY(63),  
  table_name VARBINARY(63),
```
The columns in this table are described in the next few paragraphs.

**db.** The name of the database containing the table to be replicated. You may employ either or both of the wildcards _ and % as part of the database name. Matching is similar to what is implemented for the **LIKE** operator.

**table_name.** The name of the table to be replicated. The table name may include either or both of the wildcards _ and %. Matching is similar to what is implemented for the **LIKE** operator.

**server_id.** The unique server ID of the MySQL instance (SQL node) where the table resides.

**binlog_type.** The type of binary logging to be employed. This is determined as shown in the following table:

<table>
<thead>
<tr>
<th>Value</th>
<th>Internal Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>NBT_DEFAULT</td>
<td>Use server default</td>
</tr>
<tr>
<td>1</td>
<td>NBT_NO_LOGGING</td>
<td>Do not log this table in the binary log</td>
</tr>
<tr>
<td>2</td>
<td>NBT_UPDATE_ONLY</td>
<td>Only updated attributes are logged</td>
</tr>
<tr>
<td>3</td>
<td>NBT_FULL</td>
<td>Log full row, even if not updated (MySQL server default behavior)</td>
</tr>
<tr>
<td>4</td>
<td>NBT_USE_UPDATE</td>
<td>(For generating NBT_UPDATED_ONLY_USE_UPDATE and NBT_FULL_USE_UPDATE values only—not intended for separate use)</td>
</tr>
<tr>
<td>5</td>
<td>[Not used]</td>
<td>---</td>
</tr>
<tr>
<td>6</td>
<td>NBT_UPDATED_ONLY_USE_UPDATE</td>
<td>Use updated attributes, even if values are unchanged (equal to NBT_UPDATED_ONLY</td>
</tr>
<tr>
<td>7</td>
<td>NBT_FULL_USE_UPDATE</td>
<td>Use full row, even if values are unchanged (equal to NBT_FULL</td>
</tr>
</tbody>
</table>

**conflict_fn.** The conflict resolution function to be applied. This function must be specified as one of those shown in the following list:

- NDB$OLD(column_name)
- NDB$MAX(column_name)
- NDB$MAX_DELETE_WIN()
- NDB$EPOCH() and NDB$EPOCH_TRANS()
- NDB$EPOCH_TRANS()
- NDB$EPOCH2()
- NDB$EPOCH2_TRANS()
- NULL: Indicates that conflict resolution is not to be used for the corresponding table.
These functions are described in the next few paragraphs.

**NDB$OLD(column_name).** If the value of `column_name` is the same on both the source and the replica, then the update is applied; otherwise, the update is not applied on the replica and an exception is written to the log. This is illustrated by the following pseudocode:

```sql
if (source_old_column_value == replica_current_column_value)
    apply_update();
else
    log_exception();
```

This function can be used for “same value wins” conflict resolution. This type of conflict resolution ensures that updates are not applied on the replica from the wrong source.

**Important**

The column value from the source's “before” image is used by this function.

**NDB$MAX(column_name).** If the “timestamp” column value for a given row coming from the source is higher than that on the replica, it is applied; otherwise it is not applied on the replica. This is illustrated by the following pseudocode:

```sql
if (source_new_column_value > replica_current_column_value)
    apply_update();
```

This function can be used for “greatest timestamp wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was most recently updated is the version that persists.

**Important**

The column value from the source's “after” image is used by this function.

**NDB$MAX_DELETE_WIN().** This is a variation on **NDB$MAX()**. Due to the fact that no timestamp is available for a delete operation, a delete using **NDB$MAX()** is in fact processed as **NDB$OLD**, but for some use cases, this is not optimal. For **NDB$MAX_DELETE_WIN()**, if the “timestamp” column value for a given row adding or updating an existing row coming from the source is higher than that on the replica, it is applied. However, delete operations are treated as always having the higher value. This is illustrated by the following pseudocode:

```sql
if (source_new_column_value > replica_current_column_value
    ||
    operation.type == "delete")
    apply_update();
```

This function can be used for “greatest timestamp, delete wins” conflict resolution. This type of conflict resolution ensures that, in the event of a conflict, the version of the row that was deleted or (otherwise) most recently updated is the version that persists.

**Note**

As with **NDB$MAX()**, the column value from the source's “after” image is the value used by this function.

**NDB$EPOCH() and NDB$EPOCH_TRANS().** The **NDB$EPOCH()** function tracks the order in which replicated epochs are applied on a replica cluster relative to changes originating on the replica. This relative ordering is used to determine whether changes originating on the replica are concurrent with any changes that originate locally, and are therefore potentially in conflict.

Most of what follows in the description of **NDB$EPOCH()** also applies to **NDB$EPOCH_TRANS()**. Any exceptions are noted in the text.

**NDB$EPOCH()** is asymmetric, operating on one NDB Cluster in a bidirectional replication configuration (sometimes referred to as “active-active” replication). We refer here to cluster on which it operates as
the primary, and the other as the secondary. The replica on the primary is responsible for detecting and handling conflicts, while the replica on the secondary is not involved in any conflict detection or handling.

When the replica on the primary detects conflicts, it injects events into its own binary log to compensate for these; this ensures that the secondary NDB Cluster eventually realigns itself with the primary and so keeps the primary and secondary from diverging. This compensation and realignment mechanism requires that the primary NDB Cluster always wins any conflicts with the secondary—that is, that the primary’s changes are always used rather than those from the secondary in event of a conflict. This “primary always wins” rule has the following implications:

- Operations that change data, once committed on the primary, are fully persistent and are not undone or rolled back by conflict detection and resolution.
- Data read from the primary is fully consistent. Any changes committed on the Primary (locally or from the replica) are not reverted later.
- Operations that change data on the secondary may later be reverted if the primary determines that they are in conflict.
- Individual rows read on the secondary are self-consistent at all times, each row always reflecting either a state committed by the secondary, or one committed by the primary.
- Sets of rows read on the secondary may not necessarily be consistent at a given single point in time. For NDB$EPOCH_TRANS(), this is a transient state; for NDB$EPOCH(), it can be a persistent state.
- Assuming a period of sufficient length without any conflicts, all data on the secondary NDB Cluster (eventually) becomes consistent with the primary’s data.

NDB$EPOCH() and NDB$EPOCH_TRANS() do not require any user schema modifications, or application changes to provide conflict detection. However, careful thought must be given to the schema used, and the access patterns used, to verify that the complete system behaves within specified limits.

Each of the NDB$EPOCH() and NDB$EPOCH_TRANS() functions can take an optional parameter; this is the number of bits to use to represent the lower 32 bits of the epoch, and should be set to no less than the value calculated as shown here:

\[
\text{CEIL}\left( \log_2 \left( \frac{\text{TimeBetweenGlobalCheckpoints}}{\text{TimeBetweenEpochs}} \right), 1 \right)
\]

For the default values of these configuration parameters (2000 and 100 milliseconds, respectively), this gives a value of 5 bits, so the default value (6) should be sufficient, unless other values are used for TimeBetweenGlobalCheckpoints, TimeBetweenEpochs, or both. A value that is too small can result in false positives, while one that is too large could lead to excessive wasted space in the database.

Both NDB$EPOCH() and NDB$EPOCH_TRANS() insert entries for conflicting rows into the relevant exceptions tables, provided that these tables have been defined according to the same exceptions table schema rules as described elsewhere in this section (see NDB$OLD(column_name)). You must create any exceptions table before creating the data table with which it is to be used.

As with the other conflict detection functions discussed in this section, NDB$EPOCH() and NDB$EPOCH_TRANS() are activated by including relevant entries in the mysql.ndb_replication table (see The ndb_replication system table). The roles of the primary and secondary NDB Clusters in this scenario are fully determined by mysql.ndb_replication table entries.

Because the conflict detection algorithms employed by NDB$EPOCH() and NDB$EPOCH_TRANS() are asymmetric, you must use different values for the server_id entries of the primary and secondary replicas.

A conflict between DELETE operations alone is not sufficient to trigger a conflict using NDB$EPOCH() or NDB$EPOCH_TRANS(), and the relative placement within epochs does not matter. (Bug #18459944)
Conflict detection status variables. Several status variables can be used to monitor conflict
detection. You can see how many rows have been found in conflict by NDB$EPOCH() since this replica
was last restarted from the current value of the Ndb_conflict_fn_epoch system status variable.

Ndb_conflict_fn_epoch_trans provides the number of rows that have been
found directly in conflict by NDB$EPOCH_TRANS(). Ndb_conflict_fn_epoch2 and
Ndb_conflict_fn_epoch2_trans show the number of rows found in conflict by NDB$EPOCH2()
and NDB$EPOCH2_TRANS(), respectively. The number of rows actually realigned, including those
affected due to their membership in or dependency on the same transactions as other conflicting rows,
is given by Ndb_conflict_trans_row_reject_count.

For more information, see Section 5.3.9.3, “NDB Cluster Status Variables”.

Limitations on NDB$EPOCH(). The following limitations currently apply when using NDB
$EPOCH() to perform conflict detection:

• Conflicts are detected using NDB Cluster epoch boundaries, with granularity proportional to
TimeBetweenEpochs (default: 100 milliseconds). The minimum conflict window is the minimum
time during which concurrent updates to the same data on both clusters always report a conflict. This
is always a nonzero length of time, and is roughly proportional to 2 * (latency + queueing +
TimeBetweenEpochs). This implies that—assuming the default for TimeBetweenEpochs and
ignoring any latency between clusters (as well as any queuing delays)—the minimum conflict window
size is approximately 200 milliseconds. This minimum window should be considered when looking at
expected application “race” patterns.

• Additional storage is required for tables using the NDB$EPOCH() and NDB$EPOCH_TRANS() functions; from 1 to 32 bits extra space per row is required, depending on the value passed to the
function.

• Conflicts between delete operations may result in divergence between the primary and secondary.
When a row is deleted on both clusters concurrently, the conflict can be detected, but is not
recorded, since the row is deleted. This means that further conflicts during the propagation of any
subsequent realignment operations are not detected, which can lead to divergence.

Deletes should be externally serialized, or routed to one cluster only. Alternatively, a separate row
should be updated transactionally with such deletes and any inserts that follow them, so that conflicts
can be tracked across row deletes. This may require changes in applications.

• Only two NDB Clusters in a birectional “active-active” configuration are currently supported when
using NDB$EPOCH() or NDB$EPOCH_TRANS() for conflict detection.

• Tables having BLOB or TEXT columns are not currently supported with NDB$EPOCH() or NDB
$EPOCH_TRANS().

NDB$EPOCH_TRANS(). NDB$EPOCH_TRANS() extends the NDB$EPOCH() function. Conflicts
are detected and handled in the same way using the “primary wins all” rule (see NDB$EPOCH()
and NDB$EPOCH_TRANS()) but with the extra condition that any other rows updated in the same
transaction in which the conflict occurred are also regarded as being in conflict. In other words, where
NDB$EPOCH() realigns individual conflicting rows on the secondary, NDB$EPOCH_TRANS() realigns
conflicting transactions.

In addition, any transactions which are detectably dependent on a conflicting transaction are also
regarded as being in conflict, these dependencies being determined by the contents of the secondary
cluster’s binary log. Since the binary log contains only data modification operations (inserts, updates,
and deletes), only overlapping data modifications are used to determine dependencies between
transactions.

NDB$EPOCH_TRANS() is subject to the same conditions and limitations as NDB$EPOCH(), and in
addition requires that all transaction IDs are recorded in the secondary’s binary log (the --ndb-log-
transaction-id option), which adds a variable overhead (up to 13 bytes per row). The deprecated
log_bin_use_v1_row_events system variable, which defaults to OFF, must not be set to ON with NDB$EPOCH_TRANS().

See NDB$EPOCH() and NDB$EPOCH_TRANS().

Status information. A server status variable Ndb_conflict_fn_max provides a count of the number of times that a row was not applied on the current SQL node due to “greatest timestamp wins” conflict resolution since the last time that mysqld was started.

The number of times that a row was not applied as the result of “same timestamp wins” conflict resolution on a given mysqld since it was restarted is given by the global status variable Ndb_conflict_fn_old. In addition to incrementing Ndb_conflict_fn_old, the primary key of the row that was not used is inserted into an exceptions table, as explained later in this section.

NDB$EPOCH2(). The NDB$EPOCH2() function is similar to NDB$EPOCH(), except that NDB$EPOCH2() provides for delete-delete handling with a bidirectional replication topology. In this scenario, primary and secondary roles are assigned to the two sources by setting the ndb_slave_conflict_role system variable to the appropriate value on each source (usually one of PRIMARY, SECONDARY). When this is done, modifications made by the secondary are reflected by the primary back to the secondary which then conditionally applies them.

NDB$EPOCH2_TRANS(). NDB$EPOCH2_TRANS() extends the NDB$EPOCH2() function. Conflicts are detected and handled in the same way, and assigning primary and secondary roles to the replicating clusters, but with the extra condition that any other rows updated in the same transaction in which the conflict occurred are also regarded as being in conflict. That is, NDB$EPOCH2() realigns individual conflicting rows on the secondary, while NDB$EPOCH_TRANS() realigns conflicting transactions.

Where NDB$EPOCH() and NDB$EPOCH_TRANS() use metadata that is specified per row, per last modified epoch, to determine on the primary whether an incoming replicated row change from the secondary is concurrent with a locally committed change; concurrent changes are regarded as conflicting, with subsequent exceptions table updates and realignment of the secondary. A problem arises when a row is deleted on the primary so there is no longer any last-modified epoch available to determine whether any replicated operations conflict, which means that conflicting delete operations are not detected. This can result in divergence, an example being a delete on one cluster which is concurrent with a delete and insert on the other; this why delete operations can be routed to only one cluster when using NDB$EPOCH() and NDB$EPOCH_TRANS().

NDB$EPOCH2() bypasses the issue just described—storing information about deleted rows on the PRIMARY—by ignoring any delete-delete conflict, and by avoiding any potential resultant divergence as well. This is accomplished by reflecting any operation successfully applied on and replicated from the secondary back to the secondary. On its return to the secondary, it can be used to reapply an operation on the secondary which was deleted by an operation originating from the primary.

When using NDB$EPOCH2(), you should keep in mind that the secondary applies the delete from the primary, removing the new row until it is restored by a reflected operation. In theory, the subsequent insert or update on the secondary conflicts with the delete from the primary, but in this case, we choose to ignore this and allow the secondary to “win”, in the interest of preventing divergence between the clusters. In other words, after a delete, the primary does not detect conflicts, and instead adopts the secondary’s following changes immediately. Because of this, the secondary’s state can revisit multiple previous committed states as it progresses to a final (stable) state, and some of these may be visible.

You should also be aware that reflecting all operations from the secondary back to the primary increases the size of the primary’s logbinary log, as well as demands on bandwidth, CPU usage, and disk I/O.

Application of reflected operations on the secondary depends on the state of the target row on the secondary. Whether or not reflected changes are applied on the secondary can be tracked by checking the Ndb_conflict_reflected_op_prepare_count and Ndb_conflict_reflected_op_discard_count status variables. The number
of changes applied is simply the difference between these two values (note that `Ndb_conflict_reflected_op_prepare_count` is always greater than or equal to `Ndb_conflict_reflected_op_discard_count`).

Events are applied if and only if both of the following conditions are true:

- The existence of the row—that is, whether or not it exists—is in accordance with the type of event. For delete and update operations, the row must already exist. For insert operations, the row must not exist.
- The row was last modified by the primary. It is possible that the modification was accomplished through the execution of a reflected operation.

If both of the conditions are not met, the reflected operation is discarded by the secondary.

**Conflict resolution exceptions table.** To use the `NDB$OLD()` conflict resolution function, it is also necessary to create an exceptions table corresponding to each NDB table for which this type of conflict resolution is to be employed. This is also true when using `NDB$EPOCH()` or `NDB$EPOCH_TRANS()`. The name of this table is that of the table for which conflict resolution is to be applied, with the string `$EX` appended. (For example, if the name of the original table is `mytable`, the name of the corresponding exceptions table name should be `mytable$EX`.) The syntax for creating the exceptions table is as shown here:

```sql
CREATE TABLE original_table$EX  {
    [NDB$]server_id INT UNSIGNED,
    [NDB$]source_server_id INT UNSIGNED,
    [NDB$]source_epoch BIGINT UNSIGNED,
    [NDB$]count INT UNSIGNED,
    [NDB$OP_TYPE ENUM('WRITE_ROW', 'UPDATE_ROW', 'DELETE_ROW',
                     'REFRESH_ROW', 'READ_ROW') NOT NULL,
    [NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
                        'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL,
    [NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL,
    original_table_pk_columns,
    [orig_table_column]|orig_table_column$OLD|orig_table_column$NEW,
    [additional_columns],
} ENGINE=NDB;
```

The first four columns are required. The names of the first four columns and the columns matching the original table's primary key columns are not critical; however, we suggest for reasons of clarity and consistency, that you use the names shown here for the `server_id`, `source_server_id`, `source_epoch`, and `count` columns, and that you use the same names as in the original table for the columns matching those in the original table's primary key.

If the exceptions table uses one or more of the optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` discussed later in this section, then each of the required columns must also be named using the prefix `NDB$`. If desired, you can use the `NDB$` prefix to name the required columns even if you do not define any optional columns, but in this case, all four of the required columns must be named using the prefix.

Following these columns, the columns making up the original table's primary key should be copied in the order in which they are used to define the primary key of the original table. The data types for the columns duplicating the primary key columns of the original table should be the same as (or larger than) those of the original columns. A subset of the primary key columns may be used.

The exceptions table must use the NDB storage engine. (An example that uses `NDB$OLD()` with an exceptions table is shown later in this section.)

Additional columns may optionally be defined following the copied primary key columns, but not before any of them; any such extra columns cannot be `NOT NULL`. NDB Cluster supports three additional, predefined optional columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, and `NDB$ORIG_TRANSID`, which are described in the next few paragraphs.
**NDB$OP_TYPE**: This column can be used to obtain the type of operation causing the conflict. If you use this column, define it as shown here:

```
NDB$OP_TYPE ENUM('WRITE_ROW', 'UPDATE_ROW', 'DELETE_ROW',
    'REFRESH_ROW', 'READ_ROW') NOT NULL
```

The `WRITE_ROW`, `UPDATE_ROW`, and `DELETE_ROW` operation types represent user-initiated operations. `REFRESH_ROW` operations are operations generated by conflict resolution in compensating transactions sent back to the originating cluster from the cluster that detected the conflict. `READ_ROW` operations are user-initiated read tracking operations defined with exclusive row locks.

**NDB$CFT_CAUSE**: You can define an optional column `NDB$CFT_CAUSE` which provides the cause of the registered conflict. This column, if used, is defined as shown here:

```
NDB$CFT_CAUSE ENUM('ROW_DOES_NOT_EXIST', 'ROW_ALREADY_EXISTS',
    'DATA_IN_CONFLICT', 'TRANS_IN_CONFLICT') NOT NULL
```

`ROW_DOES_NOT_EXIST` can be reported as the cause for `UPDATE_ROW` and `WRITE_ROW` operations; `ROW_ALREADY_EXISTS` can be reported for `WRITE_ROW` events. `DATA_IN_CONFLICT` is reported when a row-based conflict function detects a conflict; `TRANS_IN_CONFLICT` is reported when a transactional conflict function rejects all of the operations belonging to a complete transaction.

**NDB$ORIG_TRANSID**: The `NDB$ORIG_TRANSID` column, if used, contains the ID of the originating transaction. This column should be defined as follows:

```
NDB$ORIG_TRANSID BIGINT UNSIGNED NOT NULL
```

`NDB$ORIG_TRANSID` is a 64-bit value generated by NDB. This value can be used to correlate multiple exceptions table entries belonging to the same conflicting transaction from the same or different exceptions tables.

Additional reference columns which are not part of the original table's primary key can be named `colname$OLD` or `colname$NEW`. `colname$OLD` references old values in update and delete operations—that is, operations containing `DELETE_ROW` events. `colname$NEW` can be used to reference new values in insert and update operations—in other words, operations using `WRITE_ROW` events, `UPDATE_ROW` events, or both types of events. Where a conflicting operation does not supply a value for a given reference column that is not a primary key, the exceptions table row contains either `NULL`, or a defined default value for that column.

### Important

The `mysql.ndb_replication` table is read when a data table is set up for replication, so the row corresponding to a table to be replicated must be inserted into `mysql.ndb_replication` before the table to be replicated is created.

### Examples

The following examples assume that you have already a working NDB Cluster replication setup, as described in Section 8.5, “Preparing the NDB Cluster for Replication”, and Section 8.6, “Starting NDB Cluster Replication (Single Replication Channel)”.

**NDB$MAX() example.** Suppose you wish to enable “greatest timestamp wins” conflict resolution on table `test.t1`, using column `mycol` as the “timestamp”. This can be done using the following steps:

1. Make sure that you have started the source `mysqld` with `--ndb-log-update-as-write=OFF`.
2. On the source, perform this `INSERT` statement:

   ```sql
   INSERT INTO mysql.ndb_replication
   VALUES ('test', 't1', 0, NULL, 'NDB$MAX(mycol)');
   ```
Examples

Inserting a 0 into the `server_id` indicates that all SQL nodes accessing this table should use conflict resolution. If you want to use conflict resolution on a specific `mysql` only, use the actual server ID.

Inserting `NULL` into the `binlog_type` column has the same effect as inserting 0 (`NBT_DEFAULT`); the server default is used.

3. Create the `test.t1` table:

```sql
CREATE TABLE test.t1 (  
    mycol INT UNSIGNED,  
) ENGINE=NDB;
```

Now, when updates are performed on this table, conflict resolution is applied, and the version of the row having the greatest value for `mycol` is written to the replica.

**Note**

Other `binlog_type` options—such as `NBT_UPDATED_ONLY_USE_UPDATE`—should be used to control logging on the source using the `ndb_replication` table rather than by using command-line options.

**NDB$OLD() example.** Suppose an `NDB` table such as the one defined here is being replicated, and you wish to enable “same timestamp wins” conflict resolution for updates to this table:

```sql
CREATE TABLE test.t2  (  
    a INT UNSIGNED NOT NULL,  
    b CHAR(25) NOT NULL,  
    mycol INT UNSIGNED NOT NULL,  
    PRIMARY KEY pk (a, b)  
)   ENGINE=NDB;
```

The following steps are required, in the order shown:

1. First—and prior to creating `test.t2`—you must insert a row into the `mysql.ndb_replication` table, as shown here:

```sql
INSERT INTO mysql.ndb_replication  
VALUES ('test', 't2', 0, NULL, 'NDB$OLD(mycol)');
```

Possible values for the `binlog_type` column are shown earlier in this section. The value `'NDB$OLD(mycol)'` should be inserted into the `conflict_fn` column.

2. Create an appropriate exceptions table for `test.t2`. The table creation statement shown here includes all required columns; any additional columns must be declared following these columns, and before the definition of the table's primary key.

```sql
CREATE TABLE test.t2$EX  (  
    server_id INT UNSIGNED,  
    source_server_id INT UNSIGNED,  
    source_epoch BIGINT UNSIGNED,  
    count INT UNSIGNED,  
    a INT UNSIGNED NOT NULL,  
    b CHAR(25) NOT NULL,  
    [additional_columns],  
    PRIMARY KEY(server_id, source_server_id, source_epoch, count)  
)   ENGINE=NDB;
```

We can include additional columns for information about the type, cause, and originating transaction ID for a given conflict. We are also not required to supply matching columns for all primary key columns in the original table. This means you can create the exceptions table like this:
3. Create the table `test.t2` as shown previously.

These steps must be followed for every table for which you wish to perform conflict resolution using `NDB$OLD()`. For each such table, there must be a corresponding row in `mysql.ndb_replication`, and there must be an exceptions table in the same database as the table being replicated.

**Read conflict detection and resolution.** NDB Cluster also supports tracking of read operations, which makes it possible in circular replication setups to manage conflicts between reads of a given row in one cluster and updates or deletes of the same row in another. This example uses `employee` and `department` tables to model a scenario in which an employee is moved from one department to another on the source cluster (which we refer to hereafter as cluster `A`) while the replica cluster (hereafter `B`) updates the employee count of the employee’s former department in an interleaved transaction.

The data tables have been created using the following SQL statements:

```sql
# Employee table
CREATE TABLE employee (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    dept INT NOT NULL
) ENGINE=NDB;

# Department table
CREATE TABLE department (
    id INT PRIMARY KEY,
    name VARCHAR(2000),
    members INT
) ENGINE=NDB;
```

The contents of the two tables include the rows shown in the (partial) output of the following `SELECT` statements:

```sql
mysql> SELECT id, name, dept FROM employee;
+-------+-------+-------+
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>dept</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>998</td>
<td>Mike</td>
<td>3</td>
</tr>
<tr>
<td>999</td>
<td>Joe</td>
<td>3</td>
</tr>
<tr>
<td>1000</td>
<td>Mary</td>
<td>3</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

mysql> SELECT id, name, members FROM department;
+--------+-------+-------+
<table>
<thead>
<tr>
<th>id</th>
<th>name</th>
<th>members</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```

---

**Note**

The `NDB$` prefix is required for the four required columns since we included at least one of the columns `NDB$OP_TYPE`, `NDB$CFT_CAUSE`, or `NDB$ORIG_TRANSID` in the table definition.
We assume that we are already using an exceptions table that includes the four required columns (and these are used for this table’s primary key), the optional columns for operation type and cause, and the original table’s primary key column, created using the SQL statement shown here:

```sql
CREATE TABLE employee$EX {
    NDB$server_id INT UNSIGNED,
    NDB$source_server_id INT UNSIGNED,
    NDB$source_epoch BIGINT UNSIGNED,
    NDB$count INT UNSIGNED,
    NDB$OP_TYPE ENUM( 'WRITE_ROW','UPDATE_ROW', 'DELETE_ROW',
                     'REFRESH_ROW','READ_ROW') NOT NULL,
    NDB$CFT_CAUSE ENUM( 'ROW_DOES_NOT_EXIST',
                        'ROW_ALREADY_EXISTS',
                        'DATA_IN_CONFLICT',
                        'TRANS_IN_CONFLICT') NOT NULL,
    id INT NOT NULL,
    PRIMARY KEY(NDB$server_id, NDB$source_server_id, NDB$source_epoch, NDB$count)
} ENGINE=NDB;
```

Suppose there occur the two simultaneous transactions on the two clusters. On cluster A, we create a new department, then move employee number 999 into that department, using the following SQL statements:

```sql
BEGIN ;
INSERT  INTO department VALUES (4, "New project", 1);
UPDATE  employee SET dept = 4 WHERE id = 999;
COMMIT;
```

At the same time, on cluster B, another transaction reads from `employee`, as shown here:

```sql
BEGIN;
SELECT name FROM employee WHERE id = 999;
UPDATE department SET members = members - 1  WHERE id = 3;
commit;
```

The conflicting transactions are not normally detected by the conflict resolution mechanism, since the conflict is between a read (`SELECT`) and an update operation. You can circumvent this issue by executing `SET ndb_log_exclusive_reads = 1` on the replica cluster. Acquiring exclusive read locks in this way causes any rows read on the source to be flagged as needing conflict resolution on the replica cluster. If we enable exclusive reads in this way prior to the logging of these transactions, the read on cluster B is tracked and sent to cluster A for resolution; the conflict on the employee row is subsequently detected and the transaction on cluster B is aborted.

The conflict is registered in the exceptions table (on cluster A) as a `READ_ROW` operation (see Conflict resolution exceptions table, for a description of operation types), as shown here:

```sql
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
```

Any existing rows found in the read operation are flagged. This means that multiple rows resulting from the same conflict may be logged in the exception table, as shown by examining the effects a conflict between an update on cluster A and a read of multiple rows on cluster B from the same table in simultaneous transactions. The transaction executed on cluster A is shown here:

```sql
BEGIN;
    INSERT INTO department VALUES (4, "New project", 0);
```
Examples

```sql
UPDATE employee SET dept = 4 WHERE dept = 3;
SELECT COUNT(*) INTO @count FROM employee WHERE dept = 4;
UPDATE department SET members = @count WHERE id = 4;
COMMIT;
```

Concurrently a transaction containing the statements shown here runs on cluster B:

```sql
SET ndb_log_exclusive_reads = 1;  # Must be set if not already enabled
...
BEGIN;
  SELECT COUNT(*) INTO @count FROM employee WHERE dept = 4 FOR UPDATE;
  UPDATE department SET members = @count WHERE id = 4;
COMMIT;
```

In this case, all three rows matching the `WHERE` condition in the second transaction’s `SELECT` are read, and are thus flagged in the exceptions table, as shown here:

```
mysql> SELECT id, NDB$OP_TYPE, NDB$CFT_CAUSE FROM employee$EX;
+-------+-------------+-------------------+
| id    | NDB$OP_TYPE | NDB$CFT_CAUSE     |
|-------+-------------+-------------------|
| 998   | READ_ROW    | TRANS_IN_CONFLICT |
| 999   | READ_ROW    | TRANS_IN_CONFLICT |
| 1000  | READ_ROW    | TRANS_IN_CONFLICT |
|......  |............. |..............     |
```

Read tracking is performed on the basis of existing rows only. A read based on a given condition track conflicts only of any rows that are found and not of any rows that are inserted in an interleaved transaction. This is similar to how exclusive row locking is performed in a single instance of NDB Cluster.
Appendix A MySQL 8.0 FAQ: NDB Cluster

In the following section, we answer questions that are frequently asked about MySQL NDB Cluster and the NDB storage engine.

Questions

• A.1: Which versions of the MySQL software support NDB Cluster? Do I have to compile from source?

• A.2: What do “NDB” and “NDBCLUSTER” mean?

• A.3: What is the difference between using NDB Cluster versus using MySQL Replication?

• A.4: Do I need any special networking to run NDB Cluster? How do computers in a cluster communicate?

• A.5: How many computers do I need to run an NDB Cluster, and why?

• A.6: What do the different computers do in an NDB Cluster?

• A.7: When I run the SHOW command in the NDB Cluster management client, I see a line of output that looks like this:

```
id=2 @10.100.10.32 (Version: 8.0.24-ndb-8.0.24 Nodegroup: 0, *)
```

What does the * mean? How is this node different from the others?

• A.8: With which operating systems can I use NDB Cluster?

• A.9: What are the hardware requirements for running NDB Cluster?

• A.10: How much RAM do I need to use NDB Cluster? Is it possible to use disk memory at all?

• A.11: What file systems can I use with NDB Cluster? What about network file systems or network shares?

• A.12: Can I run NDB Cluster nodes inside virtual machines (such as those created by VMWare, VirtualBox, Parallels, or Xen)?

• A.13: I am trying to populate an NDB Cluster database. The loading process terminates prematurely and I get an error message like this one: ERROR 1114: The table 'my_cluster_table' is full Why is this happening?

• A.14: NDB Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

• A.15: Do I have to learn a new programming or query language to use NDB Cluster?

• A.16: What programming languages and APIs are supported by NDB Cluster?

• A.17: Does NDB Cluster include any management tools?

• A.18: How do I find out what an error or warning message means when using NDB Cluster?

• A.19: Is NDB Cluster transaction-safe? What isolation levels are supported?

• A.20: What storage engines are supported by NDB Cluster?

• A.21: In the event of a catastrophic failure—for example, the whole city loses power and my UPS fails—would I lose all my data?

• A.22: Is it possible to use FULLTEXT indexes with NDB Cluster?
• A.23: Can I run multiple nodes on a single computer?
• A.24: Can I add data nodes to an NDB Cluster without restarting it?
• A.25: Are there any limitations that I should be aware of when using NDB Cluster?
• A.26: Does NDB Cluster support foreign keys?
• A.27: How do I import an existing MySQL database into an NDB Cluster?
• A.28: How do NDB Cluster nodes communicate with one another?
• A.29: What is an arbitrator?
• A.30: What data types are supported by NDB Cluster?
• A.31: How do I start and stop NDB Cluster?
• A.32: What happens to NDB Cluster data when the NDB Cluster is shut down?
• A.33: Is it a good idea to have more than one management node for an NDB Cluster?
• A.34: Can I mix different kinds of hardware and operating systems in one NDB Cluster?
• A.35: Can I run two data nodes on a single host? Two SQL nodes?
• A.36: Can I use host names with NDB Cluster?
• A.37: Does NDB Cluster support IPv6?
• A.38: How do I handle MySQL users in an NDB Cluster having multiple MySQL servers?
• A.39: How do I continue to send queries in the event that one of the SQL nodes fails?
• A.40: How do I back up and restore an NDB Cluster?
• A.41: What is an “angel process”?

Questions and Answers

A.1: Which versions of the MySQL software support NDB Cluster? Do I have to compile from source?

NDB Cluster is not supported in standard MySQL Server 8.0 releases. Instead, MySQL NDB Cluster is provided as a separate product. Available NDB Cluster release series include the following:

• **NDB Cluster 7.2.** This series is no longer supported for new deployments or maintained. Users of NDB Cluster 7.2 should upgrade to a newer release series as soon as possible. We recommend that new deployments use the latest NDB Cluster 8.0 release.

• **NDB Cluster 7.3.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.3 release can be obtained from [https://dev.mysql.com/downloads/cluster/](https://dev.mysql.com/downloads/cluster/).

• **NDB Cluster 7.4.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The most recent NDB Cluster 7.4 release can be obtained from [https://dev.mysql.com/downloads/cluster/](https://dev.mysql.com/downloads/cluster/).

• **NDB Cluster 7.5.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 7.6 release. The latest NDB Cluster 7.5 releases can be obtained from [https://dev.mysql.com/downloads/cluster/](https://dev.mysql.com/downloads/cluster/).
• **NDB Cluster 7.6.** This series is a previous General Availability (GA) version of NDB Cluster, still available for production use, although we recommend that new deployments use the latest NDB Cluster 8.0 release. The latest NDB Cluster 7.6 releases can be obtained from [https://dev.mysql.com/downloads/cluster/](https://dev.mysql.com/downloads/cluster/).

• **NDB Cluster 8.0.** This series is the most recent General Availability (GA) version of NDB Cluster, based on version 8.0 of the NDB storage engine and MySQL Server 8.0. NDB Cluster 8.0 is available for production use; new deployments intended for production should use the latest GA release in this series, which is currently NDB Cluster 8.0.24. You can obtain the most recent NDB Cluster 8.0 release from [https://dev.mysql.com/downloads/cluster/](https://dev.mysql.com/downloads/cluster/). For information about new features and other important changes in this series, see *What is New in NDB Cluster*.

You can obtain and compile NDB Cluster from source (see Section 4.1.4, “Building NDB Cluster from Source on Linux”, and Section 4.2.2, “Compiling and Installing NDB Cluster from Source on Windows”), but for all but the most specialized cases, we recommend using one of the following installers provided by Oracle that is appropriate to your operating platform and circumstances:

- The web-based **NDB Cluster Auto-Installer** (works on all platforms supported by NDB)
- Linux **binary release** (*tar.gz* file)
- Linux **RPM package**
- Linux **.deb file**
- Windows **binary “no-install” release**
- Windows **MSI Installer**

Installation packages may also be available from your platform's package management system.

You can determine whether your MySQL Server has NDB support using one of the statements **SHOW VARIABLES LIKE 'have_%', SHOW ENGINES, or SHOW PLUGINS**.

**A.2: What do “NDB” and “NDBCLUSTER” mean?**

“NDB” stands for “Network Database”. NDB and NDBCLUSTER are both names for the storage engine that enables clustering support with MySQL. NDB is preferred, but either name is correct.

**A.3: What is the difference between using NDB Cluster versus using MySQL Replication?**

In traditional MySQL replication, a source MySQL server updates one or more replicas. Transactions are committed sequentially, and a slow transaction can cause the replica to lag behind the source. This means that if the source fails, it is possible that the replica might not have recorded the last few transactions. If a transaction-safe engine such as InnoDB is being used, a transaction is either completed on the replica or not applied at all, but replication does not guarantee that all data on the source and the replica remains consistent at all times. In NDB Cluster, all data nodes are kept in synchrony, and a transaction committed by any one data node is committed for all data nodes. In the event of a data node failure, all remaining data nodes remain in a consistent state.

In short, whereas standard MySQL replication is *asynchronous*, NDB Cluster is *synchronous*.

Asynchronous replication is also available in NDB Cluster. **NDB Cluster Replication** (also sometimes known as “geo-replication”) includes the capability to replicate both between two NDB Clusters, and from an NDB Cluster to a non-Cluster MySQL server. See Chapter 8, **NDB Cluster Replication**.

**A.4: Do I need any special networking to run NDB Cluster? How do computers in a cluster communicate?**

NDB Cluster is intended to be used in a high-bandwidth environment, with computers connecting using TCP/IP. Its performance depends directly upon the connection speed between the cluster's computers. The minimum connectivity requirements for NDB Cluster include a typical 100-megabit Ethernet network or the equivalent. We recommend you use gigabit Ethernet whenever available.
A.5: How many computers do I need to run an NDB Cluster, and why?

A minimum of three computers is required to run a viable cluster. However, the minimum recommended number of computers in an NDB Cluster is four: one each to run the management and SQL nodes, and two computers to serve as data nodes. The purpose of the two data nodes is to provide redundancy; the management node must run on a separate machine to guarantee continued arbitration services in the event that one of the data nodes fails.

To provide increased throughput and high availability, you should use multiple SQL nodes (MySQL Servers connected to the cluster). It is also possible (although not strictly necessary) to run multiple management servers.

A.6: What do the different computers do in an NDB Cluster?

An NDB Cluster has both a physical and logical organization, with computers being the physical elements. The logical or functional elements of a cluster are referred to as nodes, and a computer housing a cluster node is sometimes referred to as a cluster host. There are three types of nodes, each corresponding to a specific role within the cluster. These are:

- **Management node.** This node provides management services for the cluster as a whole, including startup, shutdown, backups, and configuration data for the other nodes. The management node server is implemented as the application ndb_mgmd; the management client used to control NDB Cluster is ndb_mgm. See Section 6.4, "ndb_mgmd — The NDB Cluster Management Server Daemon", and Section 6.5, "ndb_mgm — The NDB Cluster Management Client", for information about these programs.

- **Data node.** This type of node stores and replicates data. Data node functionality is handled by instances of the NDB data node process ndbd. For more information, see Section 6.1, "ndbd — The NDB Cluster Data Node Daemon".

- **SQL node.** This is simply an instance of MySQL Server (mysqld) that is built with support for the NDBCLUSTER storage engine and started with the --ndb-cluster option to enable the engine and the --ndb-connectstring option to enable it to connect to an NDB Cluster management server. For more about these options, see Section 5.3.9.1, "MySQL Server Options for NDB Cluster".

A.7: When I run the SHOW command in the NDB Cluster management client, I see a line of output that looks like this:

```
id=2  @10.100.10.32 (Version: 8.0.24-ndb-8.0.24 Nodegroup: 0, *)
```

What does the * mean? How is this node different from the others?

The simplest answer is, "It's not something you can control, and it's nothing that you need to worry about in any case, unless you're a software engineer writing or analyzing the NDB Cluster source code".

If you don't find that answer satisfactory, here's a longer and more technical version:

A number of mechanisms in NDB Cluster require distributed coordination among the data nodes. These distributed algorithms and protocols include global checkpointing, DDL (schema) changes, and node restart handling. To make this coordination simpler, the data nodes "elect" one of their number
to act as leader. There is no user-facing mechanism for influencing this selection, which is completely automatic; the fact that it is automatic is a key part of NDB Cluster's internal architecture.

When a node acts as the "leader" for any of these mechanisms, it is usually the point of coordination for the activity, and the other nodes act as "followers", carrying out their parts of the activity as directed by the leader. If the node acting as leader fails, then the remaining nodes elect a new leader. Tasks in progress that were being coordinated by the old leader may either fail or be continued by the new leader, depending on the actual mechanism involved.

It is possible for some of these different mechanisms and protocols to have different leader nodes, but in general the same leader is chosen for all of them. The node indicated as the leader in the output of SHOW in the management client is known internally as the DICT manager, responsible for coordinating DDL and metadata activity.

NDB Cluster is designed in such a way that the choice of leader has no discernible effect outside the cluster itself. For example, the current leader does not have significantly higher CPU or resource usage than the other data nodes, and failure of the leader should not have a significantly different impact on the cluster than the failure of any other data node.

A.8: With which operating systems can I use NDB Cluster?

NDB Cluster is supported on most Unix-like operating systems. NDB Cluster is also supported in production settings on Microsoft Windows operating systems.

For more detailed information concerning the level of support which is offered for NDB Cluster on various operating system versions, operating system distributions, and hardware platforms, please refer to https://www.mysql.com/support/supportedplatforms/cluster.html.

A.9: What are the hardware requirements for running NDB Cluster?

NDB Cluster should run on any platform for which NDB-enabled binaries are available. For data nodes and API nodes, faster CPUs and more memory are likely to improve performance, and 64-bit CPUs are likely to be more effective than 32-bit processors. There must be sufficient memory on machines used for data nodes to hold each node's share of the database (see How much RAM do I Need? for more information). For a computer which is used only for running the NDB Cluster management server, the requirements are minimal: a common desktop PC (or the equivalent) is generally sufficient for this task. Nodes can communicate through the standard TCP/IP network and hardware. They can also use the high-speed SCI protocol; however, special networking hardware and software are required to use SCI (see Section 5.4, "Using High-Speed Interconnects with NDB Cluster").

A.10: How much RAM do I need to use NDB Cluster? Is it possible to use disk memory at all?

NDB Cluster was originally implemented as in-memory only, but all versions currently available also provide the ability to store NDB Cluster on disk. See Section 7.10, “NDB Cluster Disk Data Tables”, for more information.

For in-memory NDB tables, you can use the following formula for obtaining a rough estimate of how much RAM is needed for each data node in the cluster:

\[
\frac{\text{Size of Database} \times \text{Number of Replicas} \times 1.1}{\text{Number of Data Nodes}}
\]

To calculate the memory requirements more exactly requires determining, for each table in the cluster database, the storage space required per row (see Data Type Storage Requirements, for details), and multiplying this by the number of rows. You must also remember to account for any column indexes as follows:

- Each primary key or hash index created for an NDBCLUSTER table requires 21–25 bytes per record. These indexes use IndexMemory.
- Each ordered index requires 10 bytes storage per record, using DataMemory.
• Creating a primary key or unique index also creates an ordered index, unless this index is created with `USING HASH`. In other words:

• A primary key or unique index on a Cluster table normally takes up 31 to 35 bytes per record.

• However, if the primary key or unique index is created with `USING HASH`, then it requires only 21 to 25 bytes per record.

Creating NDB Cluster tables with `USING HASH` for all primary keys and unique indexes generally causes table updates to run more quickly—in some cases by a much as 20 to 30 percent faster than updates on tables where `USING HASH` was not used in creating primary and unique keys. This is due to the fact that less memory is required (because no ordered indexes are created), and that less CPU must be utilized (because fewer indexes must be read and possibly updated). However, it also means that queries that could otherwise use range scans must be satisfied by other means, which can result in slower selects.

When calculating Cluster memory requirements, you may find useful the `ndb_size.pl` utility which is available in recent MySQL 8.0 releases. This Perl script connects to a current (non-Cluster) MySQL database and creates a report on how much space that database would require if it used the `NDBCLUSTER` storage engine. For more information, see Section 6.28, "`ndb_size.pl — NDBCLUSTER Size Requirement Estimator"."

It is especially important to keep in mind that every NDB Cluster table must have a primary key. The NDB storage engine creates a primary key automatically if none is defined; this primary key is created without `USING HASH`.

You can determine how much memory is being used for storage of NDB Cluster data and indexes at any given time using the `REPORT MEMORYUSAGE` command in the `ndb_mgm` client; see Section 7.1, "Commands in the NDB Cluster Management Client", for more information. In addition, warnings are written to the cluster log when 80% of available `DataMemory` or (prior to NDB 7.6) `IndexMemory` is in use, and again when usage reaches 90%, 99%, and 100%.

A.11: What file systems can I use with NDB Cluster? What about network file systems or network shares?

Generally, any file system that is native to the host operating system should work well with NDB Cluster. If you find that a given file system works particularly well (or not so especially well) with NDB Cluster, we invite you to discuss your findings in the NDB Cluster Forums.

For Windows, we recommend that you use `NTFS` file systems for NDB Cluster, just as we do for standard MySQL. We do not test NDB Cluster with `FAT` or `VFAT` file systems. Because of this, we do not recommend their use with MySQL or NDB Cluster.

NDB Cluster is implemented as a shared-nothing solution; the idea behind this is that the failure of a single piece of hardware should not cause the failure of multiple cluster nodes, or possibly even the failure of the cluster as a whole. For this reason, the use of network shares or network file systems is not supported for NDB Cluster. This also applies to shared storage devices such as SANs.

A.12: Can I run NDB Cluster nodes inside virtual machines (such as those created by VMWare, VirtualBox, Parallels, or Xen)?

NDB Cluster is supported for use in virtual machines. We currently support and test using Oracle VM.

Some NDB Cluster users have successfully deployed NDB Cluster using other virtualization products; in such cases, Oracle can provide NDB Cluster support, but issues specific to the virtual environment must be referred to that product's vendor.

A.13: I am trying to populate an NDB Cluster database. The loading process terminates prematurely and I get an error message like this one: ERROR 1114: The table 'my_cluster_table' is full. Why is this happening?
The cause is very likely to be that your setup does not provide sufficient RAM for all table data and all indexes, including the primary key required by the NDB storage engine and automatically created in the event that the table definition does not include the definition of a primary key.

It is also worth noting that all data nodes should have the same amount of RAM, since no data node in a cluster can use more memory than the least amount available to any individual data node. For example, if there are four computers hosting Cluster data nodes, and three of these have 3GB of RAM available to store Cluster data while the remaining data node has only 1GB RAM, then each data node can devote at most 1GB to NDB Cluster data and indexes.

In some cases it is possible to get Table is full errors in MySQL client applications even when ndb_mgm -e "ALL REPORT MEMORYUSAGE" shows significant free DataMemory. You can force NDB to create extra partitions for NDB Cluster tables and thus have more memory available for hash indexes by using the MAX_ROWS option for CREATE TABLE. In general, setting MAX_ROWS to twice the number of rows that you expect to store in the table should be sufficient.

For similar reasons, you can also sometimes encounter problems with data node restarts on nodes that are heavily loaded with data. The MinFreePct parameter can help with this issue by reserving a portion (5% by default) of DataMemory and (prior to NDB 7.6) IndexMemory for use in restarts. This reserved memory is not available for storing NDB tables or data.

A.14: NDB Cluster uses TCP/IP. Does this mean that I can run it over the Internet, with one or more nodes in remote locations?

It is very unlikely that a cluster would perform reliably under such conditions, as NDB Cluster was designed and implemented with the assumption that it would be run under conditions guaranteeing dedicated high-speed connectivity such as that found in a LAN setting using 100 Mbps or gigabit Ethernet—preferably the latter. We neither test nor warrant its performance using anything slower than this.

Also, it is extremely important to keep in mind that communications between the nodes in an NDB Cluster are not secure; they are neither encrypted nor safeguarded by any other protective mechanism. The most secure configuration for a cluster is in a private network behind a firewall, with no direct access to any Cluster data or management nodes from outside. (For SQL nodes, you should take the same precautions as you would with any other instance of the MySQL server.) For more information, see Section 7.17, “NDB Cluster Security Issues”.

A.15: Do I have to learn a new programming or query language to use NDB Cluster?

No. Although some specialized commands are used to manage and configure the cluster itself, only standard (My)SQL statements are required for the following operations:

- Creating, altering, and dropping tables
- Inserting, updating, and deleting table data
- Creating, changing, and dropping primary and unique indexes

Some specialized configuration parameters and files are required to set up an NDB Cluster—see Section 5.3, “NDB Cluster Configuration Files”, for information about these.

A few simple commands are used in the NDB Cluster management client (ndb_mgm) for tasks such as starting and stopping cluster nodes. See Section 7.1, “Commands in the NDB Cluster Management Client”.

A.16: What programming languages and APIs are supported by NDB Cluster?

NDB Cluster supports the same programming APIs and languages as the standard MySQL Server, including ODBC, .Net, the MySQL C API, and numerous drivers for popular scripting languages such as PHP, Perl, and Python. NDB Cluster applications written using these APIs behave similarly to other MySQL applications; they transmit SQL statements to a MySQL Server (in the case of NDB Cluster,
an SQL node), and receive responses containing rows of data. For more information about these APIs, see Connectors and APIs.

NDB Cluster also supports application programming using the NDB API, which provides a low-level C++ interface to NDB Cluster data without needing to go through a MySQL Server. See The NDB API. In addition, many NDBCLUSTER management functions are exposed by the C-language MGM API; see The MGM API, for more information.

NDB Cluster also supports Java application programming using ClusterJ, which supports a domain object model of data using sessions and transactions. See Java and NDB Cluster, for more information.

In addition, NDB Cluster provides support for memcached, allowing developers to access data stored in NDB Cluster using the memcached interface; for more information, see ndbmemcache—Memcache API for NDB Cluster (NO LONGER SUPPORTED).

NDB Cluster also includes adapters supporting NoSQL applications written against Node.js, with NDB Cluster as the data store. See MySQL NoSQL Connector for JavaScript, for more information.

A.17: Does NDB Cluster include any management tools?

NDB Cluster includes a command line client for performing basic management functions. See Section 6.5, "ndb_mgm — The NDB Cluster Management Client", and Section 7.1, "Commands in the NDB Cluster Management Client".

NDB Cluster 7.6 and earlier are also supported by MySQL Cluster Manager, a separate product providing an advanced command line interface that can automate many NDB Cluster management tasks such as rolling restarts and configuration changes. Beginning with version 1.4.8, MySQL Cluster Manager also provides experimental support for NDB Cluster 8.0. For more information about MySQL Cluster Manager, see MySQL™ Cluster Manager 1.4.8 User Manual.

NDB Cluster also provides a graphical, browser-based Auto-Installer for setting up and deploying NDB Cluster, as part of the NDB Cluster software distribution. For more information, see The NDB Cluster Auto-Installer (NDB 7.5) (No longer supported).

A.18: How do I find out what an error or warning message means when using NDB Cluster?

There are two ways in which this can be done:

- From within the mysql client, use SHOW ERRORS or SHOW WARNINGS immediately upon being notified of the error or warning condition.
- From a system shell prompt, use perror --ndb error_code.

A.19: Is NDB Cluster transaction-safe? What isolation levels are supported?

Yes. For tables created with the NDB storage engine, transactions are supported. Currently, NDB Cluster supports only the READ COMMITTED transaction isolation level.

A.20: What storage engines are supported by NDB Cluster?

NDB Cluster requires the NDB storage engine. That is, in order for a table to be shared between nodes in an NDB Cluster, the table must be created using ENGINE=NDB (or the equivalent option ENGINE=NDBCLUSTER).

It is possible to create tables using other storage engines (such as InnoDB or MyISAM) on a MySQL server being used with NDB Cluster, but since these tables do not use NDB, they do not participate in clustering; each such table is strictly local to the individual MySQL server instance on which it is created.

NDB Cluster is quite different from InnoDB clustering with regard to architecture, requirements, and implementation; despite any similarity in their names, the two are not compatible. For more information
about InnoDB clustering, see Using MySQL AdminAPI. See also Section 3.6, “MySQL Server Using InnoDB Compared with NDB Cluster”, for information about the differences between the NDB and InnoDB storage engines.

A.21: In the event of a catastrophic failure— for example, the whole city loses power and my UPS fails—would I lose all my data?

All committed transactions are logged. Therefore, although it is possible that some data could be lost in the event of a catastrophe, this should be quite limited. Data loss can be further reduced by minimizing the number of operations per transaction. (It is not a good idea to perform large numbers of operations per transaction in any case.)

A.22: Is it possible to use FULLTEXT indexes with NDB Cluster?

FULLTEXT indexing is currently supported only by the InnoDB and MyISAM storage engines. See Full-Text Search Functions, for more information.

A.23: Can I run multiple nodes on a single computer?

It is possible but not always advisable. One of the chief reasons to run a cluster is to provide redundancy. To obtain the full benefits of this redundancy, each node should reside on a separate machine. If you place multiple nodes on a single machine and that machine fails, you lose all of those nodes. For this reason, if you do run multiple data nodes on a single machine, it is extremely important that they be set up in such a way that the failure of this machine does not cause the loss of all the data nodes in a given node group.

Given that NDB Cluster can be run on commodity hardware loaded with a low-cost (or even no-cost) operating system, the expense of an extra machine or two is well worth it to safeguard mission-critical data. It also worth noting that the requirements for a cluster host running a management node are minimal. This task can be accomplished with a 300 MHz Pentium or equivalent CPU and sufficient RAM for the operating system, plus a small amount of overhead for the ndb_mgmd and ndb_mgm processes.

It is acceptable to run multiple cluster data nodes on a single host that has multiple CPUs, cores, or both. The NDB Cluster distribution also provides a multithreaded version of the data node binary intended for use on such systems. For more information, see Section 6.3, “ndbmt — The NDB Cluster Data Node Daemon (Multi-Threaded)”. It is also possible in some cases to run data nodes and SQL nodes concurrently on the same machine; how well such an arrangement performs is dependent on a number of factors such as number of cores and CPUs as well as the amount of disk and memory available to the data node and SQL node processes, and you must take these factors into account when planning such a configuration.

A.24: Can I add data nodes to an NDB Cluster without restarting it?

It is possible to add new data nodes to a running NDB Cluster without taking the cluster offline. For more information, see Section 7.7, “Adding NDB Cluster Data Nodes Online”.

For other types of NDB Cluster nodes, a rolling restart is all that is required (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”).

A.25: Are there any limitations that I should be aware of when using NDB Cluster?

Limitations on NDB tables in MySQL NDB Cluster include the following:

• Temporary tables are not supported; a CREATE TEMPORARY TABLE statement using ENGINE=NDB or ENGINE=NDBCLUSTER fails with an error.

• The only types of user-defined partitioning supported for NDBCLUSTER tables are KEY and LINEAR KEY. Trying to create an NDB table using any other partitioning type fails with an error.
- **FULLTEXT** indexes are not supported.
- Index prefixes are not supported. Only complete columns may be indexed.
- Spatial indexes are not supported (although spatial columns can be used). See Spatial Data Types.
- Support for partial transactions and partial rollbacks is comparable to that of other transactional storage engines such as InnoDB that can roll back individual statements.
- The maximum number of attributes allowed per table is 512. Attribute names cannot be any longer than 31 characters. For each table, the maximum combined length of the table and database names is 122 characters.
- Prior to NDB 8.0, the maximum size for a table row is 14 kilobytes, not counting BLOB values. In NDB 8.0, this maximum is increased to 30000 bytes. See Section 3.7.5, “Limits Associated with Database Objects in NDB Cluster”, for more information.

There is no set limit for the number of rows per NDB table. Limits on table size depend on a number of factors, in particular on the amount of RAM available to each data node.

For a complete listing of limitations in NDB Cluster, see Section 3.7, “Known Limitations of NDB Cluster”. See also Section 3.7.11, “Previous NDB Cluster Issues Resolved in NDB Cluster 8.0”.

**A.26: Does NDB Cluster support foreign keys?**

NDB Cluster provides support for foreign key constraints which is comparable to that found in the InnoDB storage engine; see FOREIGN KEY Constraints, for more detailed information, as well as FOREIGN KEY Constraints. Applications requiring foreign key support should use NDB Cluster 7.3, 7.4, 7.5, or later.

**A.27: How do I import an existing MySQL database into an NDB Cluster?**

You can import databases into NDB Cluster much as you would with any other version of MySQL. Other than the limitations mentioned elsewhere in this FAQ, the only other special requirement is that any tables to be included in the cluster must use the NDB storage engine. This means that the tables must be created with ENGINE=NDB or ENGINE=NDBCLUSTER.

It is also possible to convert existing tables that use other storage engines to NDBCLUSTER using one or more ALTER TABLE statement. However, the definition of the table must be compatible with the NDBCLUSTER storage engine prior to making the conversion. In MySQL 8.0, an additional workaround is also required; see Section 3.7, “Known Limitations of NDB Cluster”, for details.

**A.28: How do NDB Cluster nodes communicate with one another?**

Cluster nodes can communicate through any of three different transport mechanisms: TCP/IP, SHM (shared memory), and SCI (Scalable Coherent Interface). Where available, SHM is used by default between nodes residing on the same cluster host; however, this is considered experimental. SCI is a high-speed (1 gigabit per second and higher), high-availability protocol used in building scalable multi-processor systems; it requires special hardware and drivers. See Section 5.4, “Using High-Speed Interconnects with NDB Cluster”, for more about using SCI as a transport mechanism for NDB Cluster.

**A.29: What is an arbitrator?**

If one or more data nodes in a cluster fail, it is possible that not all cluster data nodes are able to “see” one another. In fact, it is possible that two sets of data nodes might become isolated from one another in a network partitioning, also known as a “split-brain” scenario. This type of situation is undesirable because each set of data nodes tries to behave as though it is the entire cluster. An arbitrator is required to decide between the competing sets of data nodes.

When all data nodes in at least one node group are alive, network partitioning is not an issue, because no single subset of the cluster can form a functional cluster on its own. The real problem arises when no single node group has all its nodes alive, in which case network partitioning (the “split-brain”)...
scenario) becomes possible. Then an arbitrator is required. All cluster nodes recognize the same node as the arbitrator, which is normally the management server; however, it is possible to configure any of the MySQL Servers in the cluster to act as the arbitrator instead. The arbitrator accepts the first set of cluster nodes to contact it, and tells the remaining set to shut down. Arbitrator selection is controlled by the ArbitrationRank configuration parameter for MySQL Server and management server nodes. You can also use the ArbitrationRank configuration parameter to control the arbitrator selection process. For more information about these parameters, see Section 5.3.5, “Defining an NDB Cluster Management Server”.

The role of arbitrator does not in and of itself impose any heavy demands upon the host so designated, and thus the arbitrator host does not need to be particularly fast or to have extra memory especially for this purpose.

A.30: What data types are supported by NDB Cluster?

NDB Cluster supports all of the usual MySQL data types, including those associated with MySQL’s spatial extensions; however, the NDB storage engine does not support spatial indexes. (Spatial indexes are supported only by MyISAM; see Spatial Data Types, for more information.) In addition, there are some differences with regard to indexes when used with NDB tables.

Note

NDB Cluster Disk Data tables (that is, tables created with TABLESPACE ... STORAGE DISK ENGINE=NDB or TABLESPACE ... STORAGE DISK ENGINE=NDBCLUSTER) have only fixed-width rows. This means that (for example) each Disk Data table record containing a VARCHAR(255) column requires space for 255 characters (as required for the character set and collation being used for the table), regardless of the actual number of characters stored therein.

See Section 3.7, “Known Limitations of NDB Cluster”, for more information about these issues.

A.31: How do I start and stop NDB Cluster?

It is necessary to start each node in the cluster separately, in the following order:

1. Start the management node, using the ndb_mgmd command. You must include the -f or --config-file option to tell the management node where its configuration file can be found.

2. Start each data node with the ndbd command. Each data node must be started with the -c or --ndb-connectstring option so that the data node knows how to connect to the management server.

3. Start each MySQL Server (SQL node) using your preferred startup script, such as mysqld_safe. Each MySQL Server must be started with the --ndbcluster and --ndb-connectstring options. These options cause mysqld to enable NDBCLUSTER storage engine support and how to connect to the management server.

Each of these commands must be run from a system shell on the machine housing the affected node. (You do not have to be physically present at the machine—a remote login shell can be used for this purpose.) You can verify that the cluster is running by starting the NDB management client ndb_mgm on the machine housing the management node and issuing the SHOW or ALL STATUS command.

To shut down a running cluster, issue the command SHUTDOWN in the management client. Alternatively, you may enter the following command in a system shell:

shell> ndb_mgm -e "SHUTDOWN"
Either of these commands causes the ndb_mgm, ndb_mgm, and any ndbd processes to terminate gracefully. MySQL servers running as SQL nodes can be stopped using mysqldadmin shutdown.

For more information, see Section 7.1, “Commands in the NDB Cluster Management Client”, and Section 4.6, “Safe Shutdown and Restart of NDB Cluster”.

MySQL Cluster Manager and the NDB Cluster Auto-Installer provide additional ways to handle starting and stopping of NDB Cluster nodes. See MySQL™ Cluster Manager 1.4.8 User Manual, and Section 4.8, “The NDB Cluster Auto-Installer (No longer supported)”, for more information about these tools.

A.32: What happens to NDB Cluster data when the NDB Cluster is shut down?

The data that was held in memory by the cluster’s data nodes is written to disk, and is reloaded into memory the next time that the cluster is started.

A.33: Is it a good idea to have more than one management node for an NDB Cluster?

It can be helpful as a fail-safe. Only one management node controls the cluster at any given time, but it is possible to configure one management node as primary, and one or more additional management nodes to take over in the event that the primary management node fails.

See Section 5.3, “NDB Cluster Configuration Files”, for information on how to configure NDB Cluster management nodes.

A.34: Can I mix different kinds of hardware and operating systems in one NDB Cluster?

Yes, as long as all machines and operating systems have the same “endianness” (all big-endian or all little-endian).

It is also possible to use software from different NDB Cluster releases on different nodes. However, we support such use only as part of a rolling upgrade procedure (see Section 7.5, “Performing a Rolling Restart of an NDB Cluster”).

A.35: Can I run two data nodes on a single host? Two SQL nodes?

Yes, it is possible to do this. In the case of multiple data nodes, it is advisable (but not required) for each node to use a different data directory. If you want to run multiple SQL nodes on one machine, each instance of mysqld must use a different TCP/IP port.

Running data nodes and SQL nodes together on the same host is possible, but you should be aware that the ndbd or ndbmtd processes may compete for memory with mysqld.

A.36: Can I use host names with NDB Cluster?

Yes, it is possible to use DNS and DHCP for cluster hosts. However, if your application requires “five nines” availability, you should use fixed (numeric) IP addresses, since making communication between Cluster hosts dependent on services such as DNS and DHCP introduces additional potential points of failure.

A.37: Does NDB Cluster support IPv6?

IPv6 is supported for connections between SQL nodes (MySQL servers), but connections between all other types of NDB Cluster nodes must use IPv4.

In practical terms, this means that you can use IPv6 for replication between NDB Clusters, but connections between nodes in the same NDB Cluster must use IPv4. For more information, see Section 8.3, “Known Issues in NDB Cluster Replication”.

(The quotation marks in this example are optional, since there are no spaces in the command string following the \-e option; in addition, the SHUTDOWN command, like other management client commands, is not case-sensitive.)
A.38: How do I handle MySQL users in an NDB Cluster having multiple MySQL servers?

MySQL user accounts and privileges are normally not automatically propagated between different MySQL servers accessing the same NDB Cluster. MySQL NDB Cluster provides support for shared and synchronized users and privileges using the `NDB_STORED_USER` privilege; see Section 7.12, “Distributed MySQL Privileges with NDB_STORED_USER”, for more information. You should be aware that this implementation is new to NDB 8.0 and is not compatible with the shared privileges mechanism employed in earlier versions of NDB Cluster, which is no longer supported in NDB 8.0.

A.39: How do I continue to send queries in the event that one of the SQL nodes fails?

MySQL NDB Cluster does not provide any sort of automatic failover between SQL nodes. Your application must be prepared to handle the loss of SQL nodes and to fail over between them.

A.40: How do I back up and restore an NDB Cluster?

You can use the NDB Cluster native backup and restore functionality in the NDB management client and the `ndb_restore` program. See Section 7.8, “Online Backup of NDB Cluster”, and Section 6.23, “ndb_restore — Restore an NDB Cluster Backup”.

You can also use the traditional functionality provided for this purpose in `mysqldump` and the MySQL server. See `mysqldump — A Database Backup Program`, for more information.

A.41: What is an “angel process”?

This process monitors and, if necessary, attempts to restart the data node process. If you check the list of active processes on your system after starting `ndbd`, you can see that there are actually 2 processes running by that name, as shown here (we omit the output from `ndb_mgmd` and `ndbd` for brevity):

```
shell> ./ndb_mgmd
shell> ps aux | grep ndb
me      23002  0.0  0.0 122948  3104 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23025  0.0  0.0   5284   820 pts/2    S+   14:14   0:00 grep ndb
shell> ./ndbd -c 127.0.0.1 --initial
shell> ps aux | grep ndb
me      23002  0.0  0.0 123080  3356 ?        Ssl  14:14   0:00 ./ndb_mgmd
me      23096  0.0  0.0  35876   812 pts/2    S+  14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23097  1.0  2.4 524116 91096 ?        Sl   14:14   0:00 ./ndbd -c 127.0.0.1 --initial
me      23168  0.0  0.0   5284   812 pts/2    R+  14:15   0:00 grep ndb
```

The `ndbd` process showing 0.0 for both memory and CPU usage is the angel process (although it actually does use a very small amount of each). This process merely checks to see if the main `ndbd` or `ndbmtd` process (the primary data node process which actually handles the data) is running. If permitted to do so (for example, if the `StopOnError` configuration parameter is set to `false`), the angel process tries to restart the primary data node process.