MySQL Cluster Administrator Guide

Information about the storage engine used in MySQL Cluster

- **NDB Cluster Concepts**
- **NDB Cluster Installation**
- **NDB Cluster Configuration**

Note that this guide is currently being re-written and that the terminology used in this guide may change.

MySQL Cluster Release 3.3.3 (alpha)
Feb. 3rd, 2004

MySQL Cluster Administrator Guide

1. MySQL Cluster Concepts

MySQL Cluster is a scalable main memory database management system with high performance and high reliability. The system consists of two parts – the MySQL Server and the NDB Cluster storage engine. This manual describes the NDB Cluster storage engine. The MySQL Server is described in the MySQL Reference Manual.

The NDB Cluster can be run integrated with the MySQL Server or as a standalone database server.

NDB Cluster synchronously replicates the database on several nodes, making the system withstand software and hardware failures without loss of information. Due to the synchronous replication algorithm, take-over of failed nodes is instantaneous.

NDB Cluster is highly configurable making it possible for the DBMS administrator to choose configuration depending on information to be stored, hardware platform to run on, and, needed performance requirements.

An easy-to-use management command-line interface makes it easy to monitor and maintain the database. A management API makes it possible to write programs that automatically monitor and maintain the database.

NDB Cluster offers two main programming interfaces: the *NDB API*, a C++ application-programming interface, provides the database programmer with a powerful way to write applications interfacing NDB Cluster directly, and the *NDB ODBC* interface can be used to interface NDB Cluster via SQL. Furthermore, when the NDB Cluster is used as a storage engine in MySQL Cluster, all the standard interfaces of MySQL becomes available for the NDB Cluster storage engine.

1.1. System Architecture

The NDB Cluster has a *shared-nothing architecture*, where each database node has its own local disk and local memory. (An exception to this is when several database nodes are executing on a symmetric multiprocessing machine. Then the database nodes communicate using shared memory.)

An *NDB Cluster database* consists of a collection of nodes of three kinds:

1. One or more *management servers* (MGM nodes)
2. One or more *database nodes* (DB nodes)
3. One or more *applications* (API nodes)

A *node* is an executing program executing on some computer hardware. By *computer,* we mean any processing hardware, including everything from processor boards to SMP machines.

The nodes can run on different computers (or the same computer) and are communicating using software modules called *transporters*. Currently, NDB Cluster supports four kinds of transporters: *TCP*, *Shared Memory* for SMP machines, *OSE Link Layer* for OSE Delta systems (a real-time operating system), and *SCI* for networks with scalable coherent interface. The SCI transporter is faster than TCP and OSE transporters, and the shared memory transporter is faster than the SCI transporter.

Management Servers (MGM)

The management servers control setup and configuration of NDB Cluster. The operation of NDB Cluster is unaffected by management server start and stop. NDB Cluster can run without any management server. It is easy to change the setup online via one or more *management*

clients that connect to the management servers. Most commonly, there is one management server and one management client running on the system.

The management server reads a *configuration file* (config.ini) with all configuration parameters at startup. The database nodes and the applications get their configuration parameters from the management server. The management server is the controller of the *cluster log* containing information about the state of the NDB Cluster. The cluster log gives a unified view of all nodes in the NDB Cluster.

Database Nodes (DB)

The database nodes consist of a collection of software modules called blocks. Some key blocks are the *transaction coordinator TC* handling the transaction connections from the applications, the *access manager ACC* handling locks and indexes, and the *tuple manager TUP* that stores database records.

The management server manages the database nodes, but the database nodes can execute without any management server when they have started.

Applications (API)

The application nodes are user applications written using *NDB API* or *ODBC.*

The user-controlled applications connect to the cluster and get their configuration from a management server. Currently all applications connect to all DB nodes (this might be changed in future releases of NDB Cluster to make it easier to handle large number of DB nodes).

1.2. Distributed Processing

Distribution and Replication

NDB Cluster automatically distributes and replicates data, making it transparent for the application how it is replicated and where it is stored. The number of replicas is configurable.

The application program interface is network transparent, meaning that applications connect to the NDB Cluster as a whole without specifying database nodes. If a database node fails, the NDB API automatically selects another database node to execute transactions for the application.

Transparent Distribution using a Hash Function

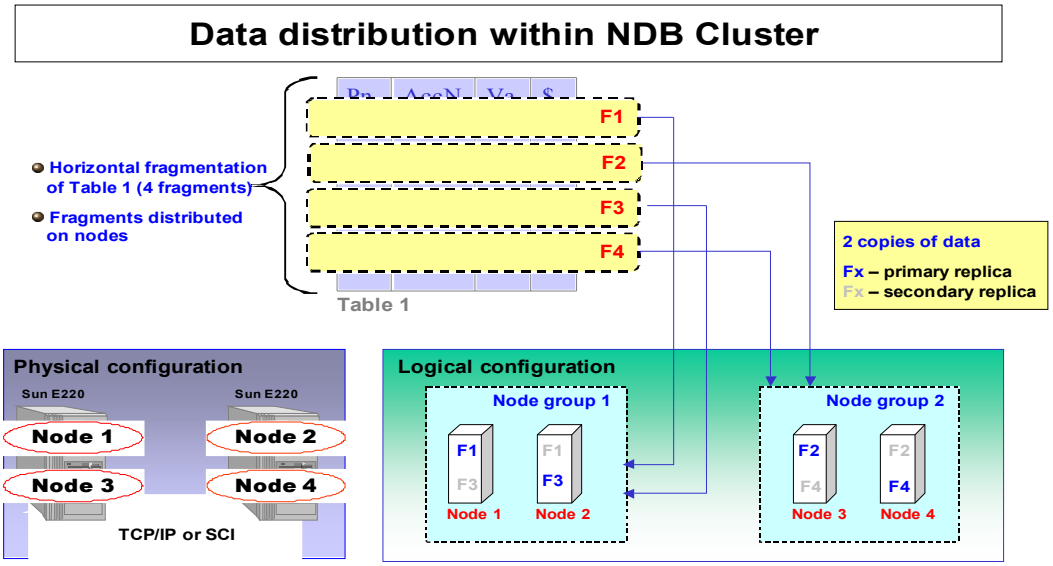
The database stores relational database tables consisting of records (also called *tuples*) of values. The tables are horizontally fragmented and the fragments are stored on different DB nodes.

NDB Cluster automatically fragments the data using a specially designed algorithm. The algorithm distributes records on nodes using a hashing key for each record; usually the hashing key is the same as the primary key.

Transparent Synchronous Replication

NDB Cluster normally replicates all data to increase reliability. The number of replicas (i.e. copies of each table) is configurable and can be one up to four. Using multiple replicas makes it possible to store records on several computers, making NDB Cluster available even during software and hardware failures. A *node group* is a set of DB nodes storing the same set of fragments. The number of nodes in any node group is the same as the number of replicas.

A typical configuration with four DB nodes and two replicas would consist of two nodes groups, each consisting of two DB nodes.



The replicas are updated synchronously, meaning that each updating transaction modifies all replicas before it is committed. This ensures that fail-over time for node failure is very short. For any failing node, any other node in the node group is immediately ready to take over transactions of the failing node. Transactions started, but not committed on a failing node are aborted and the applications need to re-execute them.

Example

Consider an NDB Cluster with four database nodes: DB1, DB2, DB3 and DB4. Then each table stored in NDB Cluster is split into four table fragments. (Each table is actually split into two times the number database nodes of the NDB Cluster for technical reasons, but we simplify it in the example to make it easier to follow). Lets denote the fragments by F1, F2, F3, and F4.

We assume that NDB Cluster has been configured to use two system replicas (i.e. all information in the database should exists in two copies). (The number of replicas must be greater or equal to the number of database nodes and less than or equal to four.)

The four fragments are thus copied into eight fragment replicas (two times the number of database nodes times the number of system replicas) and the fragment replicas are stored as follows:

- Database node DB1 stores fragment replicas F1, F3
- Database node DB2 stores fragment replicas F1, F3
- Database node DB3 stores fragment replicas F2, F4
- Database node DB4 stores fragment replicas F2, F4

Each database node is primary for one fragment.

Concurrency and consistency

NDB Cluster uses pessimistic concurrency control based on locking. If it is impossible to attain a requested lock (implicit and depending on database operation) within a specified time, then a timeout error occurs.

NDB Cluster provides concurrency control through locks on database records. The locks ensure that multiple transactions do not modify the same record at the same time. Deadlocks are handled using timeouts and lock queues.

Consistency is ensured through shadow copies in memory. A rollback sets the current value to the old value.

1.3. Cluster Nodes (DB, MGM, and API)

Management Server (MGM)

The management server creates the cluster log, which is the main source of information about the cluster. The cluster log typically contains entries like:

2002-05-06 07:39:36 [MgmSrvr] ALERT -- Node 2: Communication to Node 3 opened

The entry begins with *date and time*, then *reporting program*, *type of entry*, *reporting node number*, and finally *log message*. There are five types of entries:

- *Alert*: Serious problem that needs the attention of the Database Administrator.
- *Critical*: System resource is in critical state, low on memory, etc.
- *Error*: System error. Something is not properly working.
- *Warning*: Information only, not an error.
- *Info*: Information about system state, etc.

Database Node (DB)

Each database node needs its own disk space or *file system*. The location of this is usually set in the configuration file (config.ini). The file system for a node consists of 11 directories of information D1-D11. The directories store all information that the node needs, including data, configuration and system files.

Be careful to allocate enough memory for the file system of each database node of the cluster.

An error log, usually called error.log contains information about local node failures. A typical entry looks like this:

```
Date/Time: Wednesday 5 June 2002 - 14:18:36
Type of error: error
Message: Path set in "NDB_FILESYSTEM" is not valid
Fault ID: 2805
Problem data:
Object of reference:  Filename::init()
ProgramName: NDB_Kernel
ProcessID: 348
TraceFile: NDB_TraceFile_3.trace
***EOM***
```

The most important fields in this, except for the date and the message, is the *Fault ID*, which is the error code, and the trace file which gives the filename of the error trace file.

Trace files describe what has happened from the perspective of a database node. These files are stored in the working directory of the node. The files are very useful for debugging the system and are used by MySQL support to find out what has happened.

Applications (API)

Load balancing is ensured by automatically selecting database nodes in a round-robin fashion. Each node is picked some number of times before going to the next. If the database node is down, then next is picked. Nodes are picked at the time when the transaction starts. The transaction is aimed towards one TC/node.

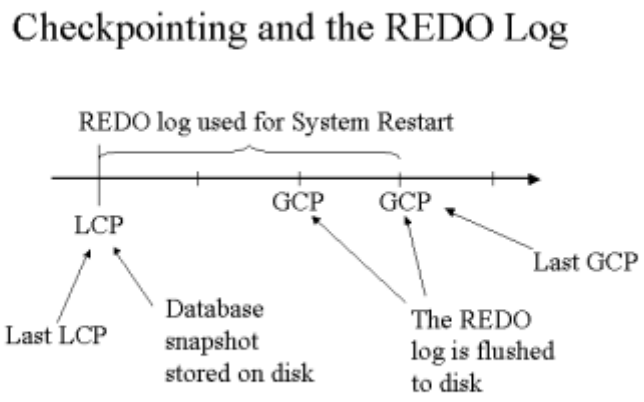
1.4. Node and System Recovery

Logging

All transactions are logged in a log called the *REDO log*. The log is stored both in main memory and on disk.

Each committed transaction gets a *global checkpoint identifier* GCI. *Global check pointing*, GCP synchronizes the main memory log with the log stored on disk. This copies information about all transaction with a GCI lower than number of the GCP to the disk making them committed to disk. In database literature, this is often called group commit.

Local check pointing is used to reduce the size of the REDO log. Local checkpoints flush the memory of the NDB node to disk.



Temporary tables

A *temporary table* is a table that is not logged. Temporary tables are more efficient since they are not part of any local or global checkpoint. Temporary tables are not restored in case of system failures.

Node and System Recovery

Heartbeat messages are used to detect if nodes (both database and application nodes) are dead. A node missing three heartbeats is perceived as dead by the rest of the NDB Cluster.

A failed node, e.g. due to heartbeat failures, can be restarted by executing the **ndb** program. The node is then recovered using the Node Recovery protocol.

Automatic Node Recovery

A node that fails (due to software or hardware failure) can be recovered using a node-recovering scheme. In principle, the node reads its LCP files and executes its REDO LOG to get close to the time when it failed. Then it asks other nodes in the same node group for additional information that is transferred to the recovering node.

In NDB Cluster, a take-over by a backup node is instantaneous. This is because all operations are replicated to the backup node. In some other DBMS, the backup node is only storing the log (corresponding to our REDO log), and in this case it takes much longer to do a take-over since the log needs to be applied.

Note that a node failure aborts some transactions even though the node is later recovered.

Automatic Multi-Node Recovery

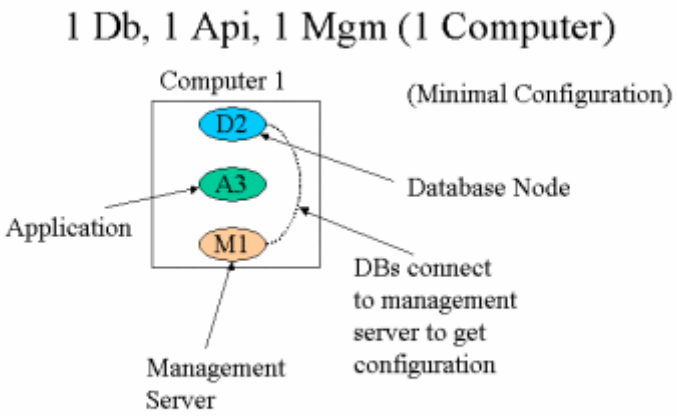
Multi-node recovery is performed in the same way as single node recovery. The NDB Cluster automatically schedules node recovery of one node at the time.

System Recovery

When all nodes of the NDB Cluster are down and later restarted, then a *system recovery* protocol uses the LCP files and the REDO log to recover database to the time of the last GCP before the system went down.

Note that information in temporary tables is not recovered during a system recovery.

1.5. Example Configuration: Single Node on a Single Computer



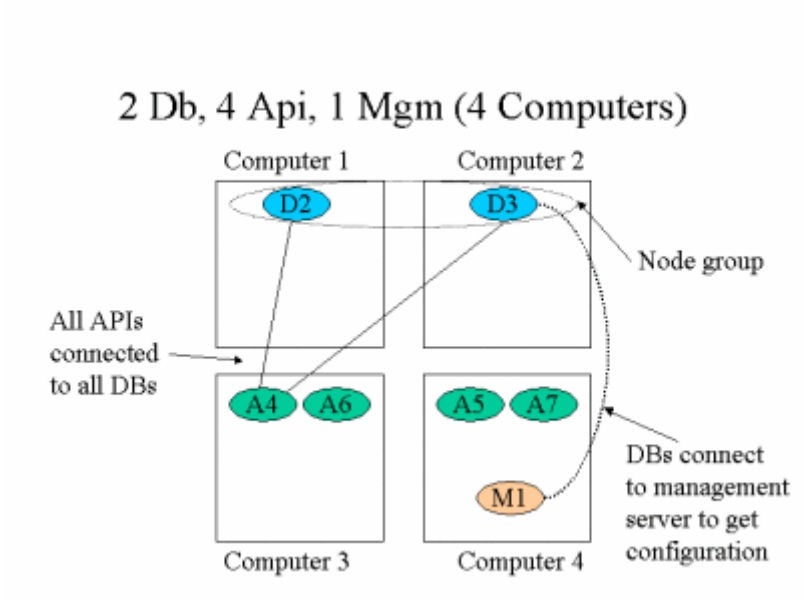
In the simplest configuration of NDB Cluster, there is one database node (DB), one management server (MGM), and one application (API). All nodes may run on the same computer.

When the database node starts, it gets its configuration from the management server.

Failure Scenarios

If there are any hardware or software failures on either the computer or the database node, a *system restart* is required.

1.6. Example Configuration: Two Nodes on Four Computers



To make the configuration more complex, we can add more applications. In the example above, we have four application programs running, A4 to A7.

In the example, we have located the applications on two separate computers. This is often done to get applications that are more effective. The applications could also have been co-located on the same computers as the DB nodes.

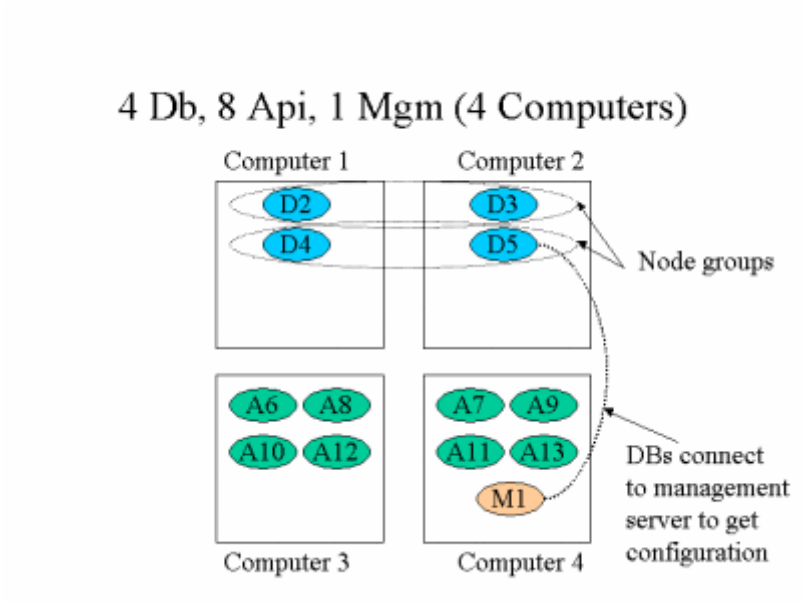
To get higher reliability, NDB Cluster may be configured to store multiple replicas on possibly different computers. In the example above, the DB nodes D2 and D3 store the same information. In case of a crash of one of them, the other will still be alive. All DB nodes that store the same information belong to the same *node group*.

Failure Scenarios

We study some failure scenarios. For this analysis, we assume that an *arbitrator* is defined and located on the management server.

Failed Nodes/Computers	Recovery Procedure
{D2}, {D3}	Node Recovery
{D2, D3}	System Restart
{C1}, {C2}	Node Recovery (after computer is replaced/restarted)
{C3}, {C4}, {C3, C4}	Restart Applications and Management Server

1.7. Example Configuration: Four Nodes on Four Computers



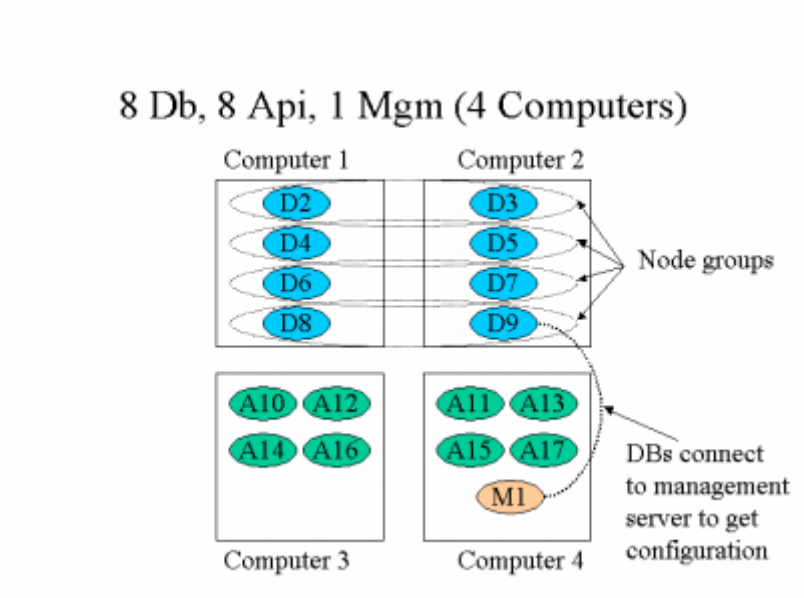
Failure Scenarios

Failed Nodes/Computers	Recovery Procedure
{D2}, {D3}, {D2, D4}, {D2, D5}, {D3, D4}, {D3, D5}	Node Recovery
{D2, D3}, {D4, D5} + all three node failures	System Restart
{C1}, {C2}	Node Recovery (after computer is replaced/restarted)
{C3}, {C4}, {C3, C4}	Restart Applications and Management Server

1.8. Example Configuration: Eight Nodes on Four Computers

Consider a configuration of the system with totally 17 nodes running on four host computers: eight database nodes, 8 applications and 1 management server.

The database nodes are usually constantly running on the hosts, but the application programs might start and stop arbitrarily. When an application program starts, it connects to every database node.



Failure Scenarios

If any of the database nodes D2-D9 gets a hardware or software error, NDB Cluster will start the node recovery protocol to restart the failed nodes. Except for when a whole node group fails, NDB Cluster can handle all two-node failures.

Failed Nodes/Computers	Recovery Procedure
{D2}, {D3}, ..., {D9}	Node Recovery
{D2,D3}, {D4,D5}, {D6,D7}, {D8,D9}	System Restart
{D2,D4}, {D2,D5}, {D2,D6}, {D2,D7}, {D2,D8}, {D2,D9}	Node Recovery
{D3,D4}, {D3,D5}, {D3,D6}, {D3,D7}, {D3,D8}, {D3,D9}	Node Recovery
{D4,D6}, {D4,D7}, {D4,D8}, {D4,D9}	Node Recovery
{D5,D6}, {D5,D7}, {D4,D8}, {D4,D9}	Node Recovery
{D6,D8}, {D6,D9}	Node Recovery
{D7,D8}, {D7,D9}	Node Recovery
{C1}, {C2}	Node Recovery if Arbitrator survived
{C1}, {C2}	System Restart if Arbitrator did not survive
{C3}, {C4}, {C3, C4}	Restart Applications and Management Server

2. NDB Cluster Installation

NDB Cluster consists of database nodes (denoted DB), management server nodes (denoted MGM), and user application programs (denoted API). The nodes are executing programs that run on possibly different computers.

The management servers are the administrative nodes of NDB Cluster. They handle all configuration parameter settings for all nodes. Each database and application node connects to a management server at startup time to get its configuration parameters.

As other nodes depend on a management server for their configuration data, a management server must be started before any other kind of node in the cluster.

Before a node can connect to a management server, it must have information about three things: (1) its own node id, (2) the name or, alternatively, the IP address of a host running a management server, and (3) the port number on which the chosen management server accepts connections from DB and API nodes. This information is stored in the nodes' *local configuration files*. Each management server also has a local configuration file.

The rest of this chapter explains how to do install NDB Cluster.

2.1. Step 1 – (Only for Windows 2000) Operating System Settings

The following steps are integrated into the Windows NDB Cluster install program. Normally the user is ***not*** required to perform these steps manually, but the steps are described here anyway in case the user chooses not to enable the settings during installation.

1. **Enable Address Windowing Extensions (AWE) Memory**
    1. Locate the file boot.ini on the boot hard drive (usually C:). If the boot.ini file cannot be found in C:\, select "Show hidden files and folders" from the folder menu (Tools, Folder Options, View, "Show hidden files and folders") and make sure the "Hide operating system files" is NOT selected.
    2. In section [Operating Systems] add **/3GB** to the operating system configuration line

Note: In rare circumstances, the /3GB flag may cause Windows to fail to start. It is recommend that this flag is *not* set if the extra memory is not needed.

2. **Enable Lock Pages in Memory**
    1. Select *Administrative Tools* in *Control Panel*
    2. Select *Local Security Policy*
    3. Select *Local Policies*
    4. Select *User Rights Assignment*
    5. Enable *Lock Pages in Memory* for the user running NDB Cluster

Note: Note that lock pages in memory *must* be enabled on Windows 2000 for NDB Cluster to operate.

2.2. Step 2 – Verify system requirements

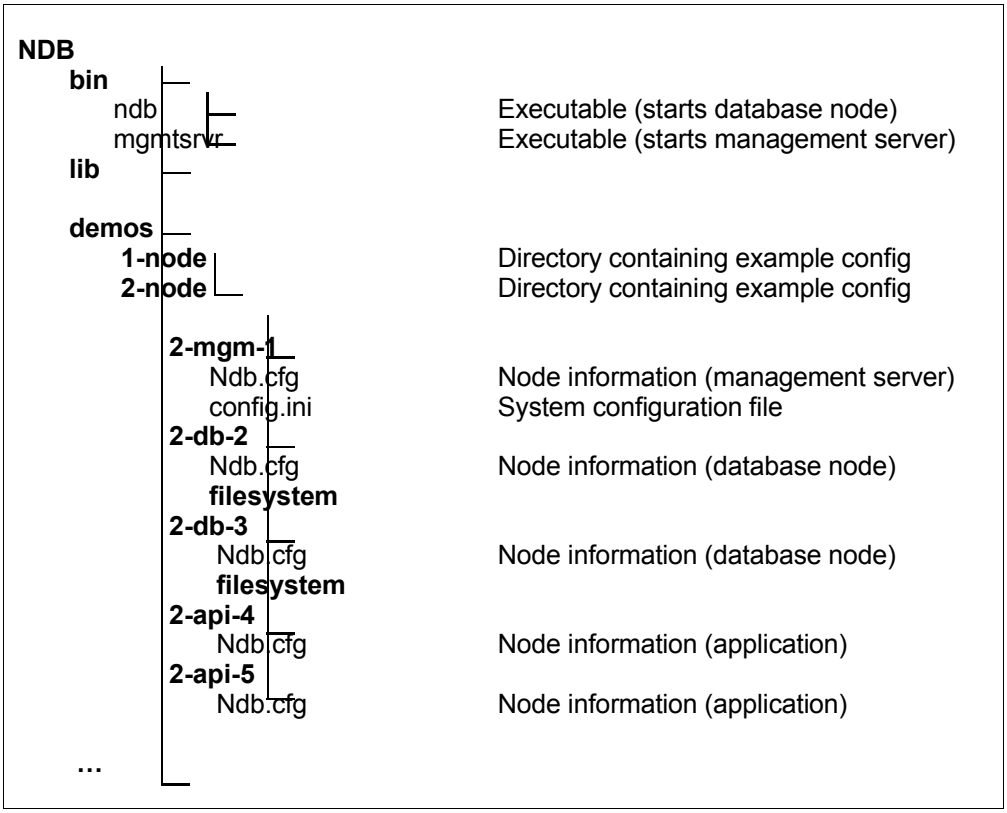
Read the *NDB Cluster Release Notes* and make sure that your computers satisfy the system requirements. Especially verify that there is enough main memory and disk space available.

2.3. Step 3 – Create directories and copy files (for each computer)

NDB Cluster come with a preformatted directory structure with binaries, example programs and demo configurations. Below we describe what such a structure may look like. Read the NDB Cluster Release Notes for more exact information.

The **bin** directory contains the executable programs, the **lib** directory contains library files used by the application programs (i.e. API nodes).

In the **demos** directory there are two example configurations, **1-node** and **2-node**. The **2-node** configuration consists of one management server (MGM), two database nodes (DB) and two applications (API).



If NDB Cluster executes on multiple computers, then each computer should have the binary executable files in **NDB/bin**. There should also be directories **NDB/demos/X-node/X-db-Y** and **NDB/demos/X-node/X-api-Z** for each database node numbered Y and each application node numbered Z.

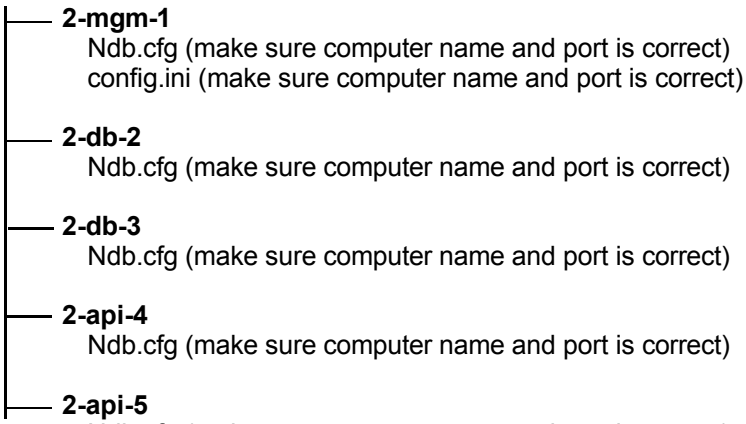
2.4. Step 4 - Create file system for each database node

Each database node needs a directory (an *NDB Cluster file system*) to store its data. Make sure that there is enough space available on the hard disk where this directory is located. In the example, the two NDB Cluster file systems for the database nodes are located in **demos/2-node/2-db-2** and **demo/2-node/2-db-3**. The file system directories must be specified with the **FileSystemPath** parameter in the configuration file (config.ini) for the NDB Cluster.

2.5. Step 5 – Create and edit configuration files

For any configuration except for the demo configurations, the configurations files should be created and edited. We describe the example with two database nodes below.

2-node



In this example, we are running on a single host computer, which we have named with two different ids. This makes it easier to modify and use the same configuration on multiple computers.

Each section of the configuration file defines a component (Computer, Node, Connection, etc) of the NDB Cluster. Parameters listed in sections named default (e.g. [DB DEFAULT]) are propagated to all components of the appropriate type (e.g. all DB nodes).

File name	Content

2-mgm-1/config.ini

```
####################################################################
#
#
#  System configuration file for NDB Cluster v2.00  (Demo 2)
#
#  MySQL AB
#  Web: www.mysql.com
#
####################################################################
#

[COMPUTER]
Id: 1
ByteOrder: Little
HostName: localhost

[COMPUTER]
Id: 2
ByteOrder: Little
HostName: localhost

[MGM]
Id: 1
ExecuteOnComputer: 1
PortNumber: 10000
PortNumberStats: 10001
ArbitrationRank: 1

[DB DEFAULT]
NoOfReplicas: 2
LockPagesInMainMemory: N
StopOnError: Y
MaxNoOfConcurrentOperations: 1024
MaxNoOfConcurrentTransactions: 1024
NoOfIndexPages: 200
NoOfDataPages: 600
TimeBetweenLocalCheckpoints: 20
TimeBetweenGlobalCheckpoints: 1500
NoOfFragmentLogFiles: 8

[DB]
Id: 2
ExecuteOnComputer: 1
FileSystemPath: WRITE_PATH_TO_FILESYSTEM_2_HERE

[DB]
Id: 3
ExecuteOnComputer: 2
FileSystemPath: WRITE_PATH_TO_FILESYSTEM_3_HERE

[API DEFAULT]
ArbitrationRank: 1

[API]
Id: 4
ExecuteOnComputer: 1

[API]
Id: 5
ExecuteOnComputer: 1

[API]
Id: 6
ExecuteOnComputer: 2

[API]
Id: 7
```

2-mgm-1/Ndb.cfg	OwnProcessId 1 127.0.0.1 10000
2-db-2/Ndb.cfg	OwnProcessId 2 127.0.0.1 10000
2-db-3/Ndb.cfg	OwnProcessId 3 127.0.0.1 10000
2-api-4/Ndb.cfg	OwnProcessId 4 127.0.0.1 10000
2-api-5/Ndb.cfg	OwnProcessId 5 127.0.0.1 10000
2-api-6/Ndb.cfg	OwnProcessId 6 127.0.0.1 10000
2-api-7/Ndb.cfg	OwnProcessId 7 127.0.0.1 10000

2.6. Step 6 – Start Management server

The management server should be started in the MGM node directory (**2-node/2-mgm-1**) by executing the command:

mgmtsrvr -c config.ini

2.7. Step 7 – Start DB nodes

From respective database node directory (e.g. 2-db-3 or 2-db-3) either type:

- **ndb** To start a DB node, or
- **ndb –i** Delete the file system and start a DB node (This is used to start a database node from scratch.)
- **ndb –n** This starts the DB node in a halted mode. That is, before running the startup protocol, the DB node halts until it receives a start signal from the management server. Querying the status of the node from the management server will return a "NOT STARTED" status. To start the node, execute the "<*id*> **start**" management command. Starting a node with "-n" is useful if you want to synchronize or control when DB nodes in the cluster starts up.

Note: All database nodes should be started with the same command and flags, i.e. either start all nodes with '**ndb**' or with '**ndb –i**'.

A node that has been stopped can be restarted with either '**ndb**' (which means that it will read its data from its file system) or with '**ndb -i**' which means that the node gets its data from the other database nodes in the NDB Cluster.

2.8. Step 8 – Start API nodes

Application programs (API nodes) should be started in their directories (e.g. **2-node/2-api-5**). Note that only four application programs can run simultaneously in the example configuration since only four API nodes are specified in the example configuration file (config.ini).

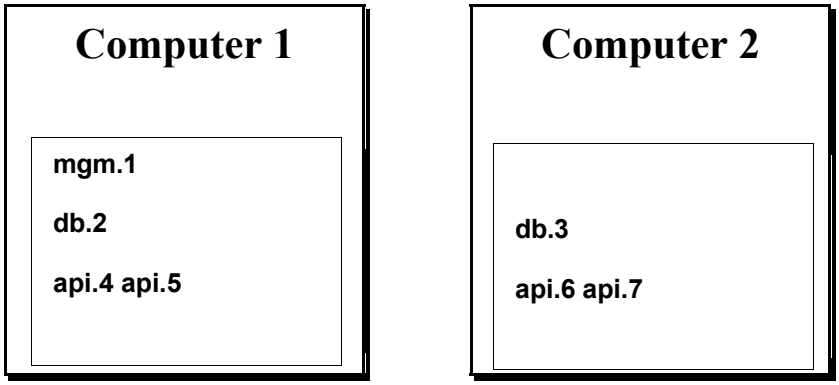
2.9. Step 9 – NDB Cluster shutdown

1. Stop database nodes by typing **all stop** in the management server command-line interface.
2. Stop the management server by typing **exit**.

2.10. Step 10 – (Optional) NDB Cluster on multiple computers

To execute NDB Cluster on multiple computers, the example configuration above can be modified in two steps:

1. Change the lines 'HostName: localhost' in the configuration file (config.ini) for the two computers to the actual names of the computers to execute on.
2. For each local configuration file (Ndb.cfg) change the line ('127.0.0.1 10000') with information about where the management server is located (IP address and port number) to the real location of the management server.



3. NDB Cluster Configuration

NDB cluster consists of nodes of three kinds - database nodes (DB), management servers (MGM), and user application programs (API).

The management server is the primary component responsible for the NDB Cluster configuration. It controls configuration parameters for all nodes of the NDB Cluster. All cluster nodes (DB, MGM or API) starts with connecting to the management server to obtain configuration information.

Before a NDB Cluster node (DB, API or MGM) node can connect to a management server and get its configuration, it needs the following information:

1. its own *node id*,
2. *computer name* or, alternatively, the *computer IP address* of a host computer running a management server, and
3. *port number* of the management server.

This information is stored in the nodes *local configuration files* (Ndb.cfg). The management server itself also receives its settings through a local configuration file (in case only one management server is used, the computer name and port number specified in the file should be the management servers own computer name and port number).

As NDB Cluster nodes depend on a management server for its configuration, a management server should be started before any other node in the cluster.

A number of parameters in the cluster can only be set before the NDB Cluster is started.

Fig.1 below illustrates NDB Cluster configuration setup for an NDB Cluster with one management server (MGM), four database nodes (DB) and two application programs (API).

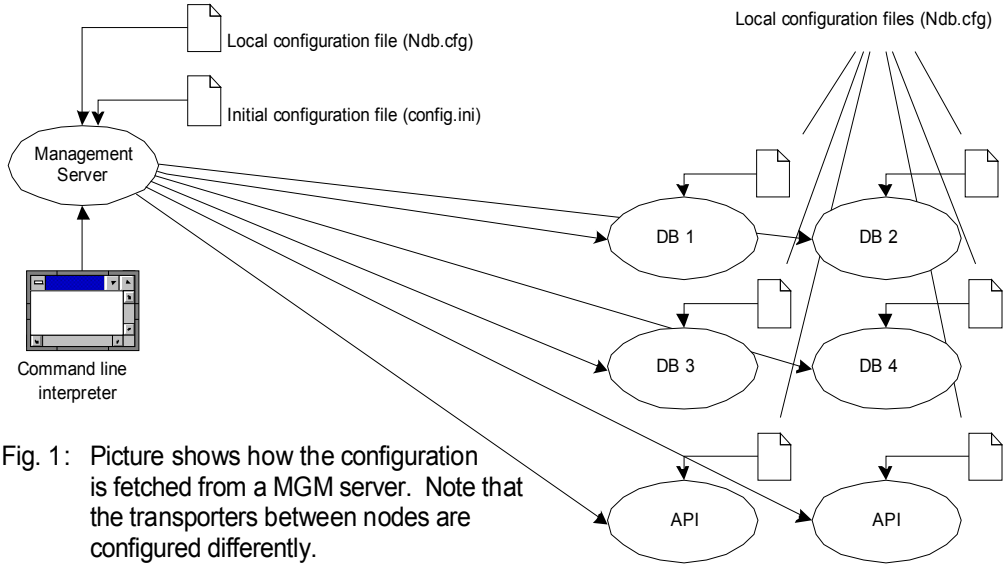


Fig. 1: Picture shows how the configuration is fetched from a MGM server. Note that the transporters between nodes are configured differently.

3.1. Environment variables (UNIX operating systems only)

Before starting an NDB cluster, environment variables need to be set for each DB and MGM node:

LD_LIBRARY_PATH	Specifies path to NDB Cluster runtime libraries. This variable is used by the management server and the API nodes.

3.2. Configuration files

A configuration file is a plain text file that contains directives for each valid parameter of the nodes. It is read and processed when the node is started.

Directives in configuration files are placed on separate lines and must be terminated with `newline`. Each directive contains a parameter identifier and its value. Both the parameter identifier and its value are case-sensitive.

Comments in configuration files must be placed on separate lines and preceded by **#**. Comments may not be included on the same lines as directives.

Updating parameters

After starting the cluster, some of the configuration parameters can be updated. This is commented for each parameter in the parameter list below.

3.3. Management Server

The management server is a stand-alone executable (node) that is started before any other cluster node. When started for the first time, it requires two configuration files: the *local configuration file* that contains its node parameters and the *configuration file* containing configuration information for the whole NDB Cluster.

3.4. Management Server Startup options

The Management Server is started as follows:

mgmtsrvr -c *ConfigFile* [**–l** *LocalConfigFile*] [-d] [-f]

where:

ConfigFile	Name of configuration file with NDB Cluster configuration data required when the server runs. Directives found in this file are described below. The configuration file can be over-written by the management server if any configuration parameter is changed. The file is typically called config.ini.
LocalConfigFile	Name of local configuration file (default name is Ndb.cfg). The local configuration file for the Management Server contains the same information as the configuration files used by DB and API nodes as described in *Configuration of DB and API nodes.*
-d	Daemon mode – starts Management Server without the command line interface. Useful when the Management Server is started automatically at boot.
-f	Fork – puts the Management Server in the background, detached from the controlling tty. Implies –d.

3.5. Management Server configuration file (config.ini)

The directives in the configuration file (config.ini) are grouped in sections. There are currently nine different types of sections:

- [SYSTEM] Parameters for whole NDB Cluster
- [COMPUTER] Parameters for host computers
- [DB], [API], [MGM] Parameters for NDB Cluster nodes
- [TCP], [SCI], [SHM], [OSE] Parameters for connections (transporters)

It is possible to omit a parameter in the configuration file if the parameter has a default value. Values for parameters without default values must be specified.

For each section type it is possible to have a default section. Values in the default section are then propagated to all later section of that type. In the example below, the parameter value for NoOfReplicas is propagated to all DB nodes (DB 2 and DB 3), thus eliminating the need to specifying the parameter for all DB sections.

```
[DB DEFAULT]
NoOfReplicas: 2

[DB]
Id: 2
ExecuteOnComputer: 1

[DB]
Id: 3
ExecuteOnComputer: 2
```

Below we list some of the most common parameters used in the configuration file (config.ini).

Note: Changing the value of a configuration parameter is a delicate process. If the parameter is changed in the wrong way, the database may become corrupt.
Always restart the management server to reload the configuration after change.

[SYSTEM] Configuration parameters for a whole NDB Cluster

Parameter: [SYSTEM] **PrimaryMGMNode**
Description: Sets primary management node where configuration changes will be synchronized.
Unit: N/A
Value range: Valid management node id
Default value: The first management node
Can be updated: Yes, after system restart

Parameter: [SYSTEM] **ConfigGenerationNumber**
Description: Whenever the management server has changed its configuration, this number is incremented. The number is used when communicating configuration changes with other MGM, API and DB nodes.
Unit: N/A
Value range: Integer >= 0
Default value: 0
Can be updated: Updated automatically when necessary

[COMPUTER] Parameters for host computers

Parameter: [COMPUTER] **Id**
Description: Number that uniquely identifies a computer.

Unit: N/A
Value range: ≥1
Default value: None
Can be updated: No

Parameter: [COMPUTER] **ByteOrder**
Description: Machine specific byte order (big or little endian).
Unit: Not currently used
Value range: "Big"/"Little"
Default value: None
Can be updated: No

Parameter: [COMPUTER] **HostName**
Description: Host name or IP-address.
Unit: N/A
Value range: Valid host name
Default value: None
Can be updated: Yes, after system restart

Parameter: [COMPUTER] **ScId1**
Description: SCI node id of SCI adapter 0.
Unit: N/A
Value range: Valid SCI node id
Default value: None
Can be updated: Yes, after system restart

Parameter: [COMPUTER] **ScId2**
Description: SCI node id of SCI adapter 1.
Unit: N/A
Value range: Valid SCI node id
Default value: None
Can be updated: Yes, after system restart

[DB] Parameters for database nodes

Parameter: [DB] **NoOfReplicas**
Description: Number of replicas to be stored in NDB Cluster.
Unit: Replicas
Value range: 1 - 4
Default value: None
Can be updated: No

Parameter: [DB] **MaxNoOfAttributes**
Description: Maximum number of attributes to be stored in NDB Cluster (summed over all tables that should be stored in NDB Cluster).

Unit: Attributes
Value range: 32 - 4096
Default value: 1000
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfTables**
Description: Maximum number of tables (including 1 system table) to be stored in NDB Cluster.
Unit: Tables
Value range: 8 - 128
Default value: 32
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfIndexes**
Description: Maximum number of indexes that can be defined in NDB Cluster Note that this parameter can never be less than MaxNoOfTables.
Unit: Indexes
Value range: 0 - 2048
Default value: 128
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOTriggers**
Description: Maximum number of triggers that can be defined in NDB Cluster Note that each index requires three triggers and each table needs three triggers during a backup.
Unit: Triggers
Value range: 0 - 2432
Default value: 768
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfSavedMessages**
Description: Maximum number of error messages stored in error log file. It also sets the maximum number of trace files saved.
Unit: N/A
Value range: ≥1
Default value: 25
Can be updated: Yes

Parameter: [DB] **LockPagesInMainMemory**
Description: Specifies whether or not the DB node will lock pages in memory.
Unit: N/A
Value range: Y/N
Default value: N
Can be updated: Yes

Parameter: [DB] **TimeBetweenWatchDogCheck**
Description: Sets the time between watchdog checks in the DB node. The watchdog is used to detect if a node is down. Missing three watchdog checks leads to termination of the node. Possible reasons for missing

watchdog checks are e.g. API program errors resulting in a loop, lengthy database operations, or slow file system performance.

Unit: Milliseconds

Value range: ≥70
Default value: 4000
Can be updated: Yes

Parameter: [DB] **StopOnError**
Description: Instructs node to stop instead of restarting if a failure occurs. If set to NO, the nodes are automatically restarted and recovered.

Unit: N/A

Value range: Y/N
Default value: Y
Can be updated: Yes

Parameter: [DB] **MaxNoOfConcurrentOperations**
Description: Maximum number of operations that can be concurrently active on all DB nodes. E.g. this parameter has do be greater than 100 to be able to run 10 parallel transactions, each containing 10 operations, on one DB node.

Unit: Operations

Value range: MaxNoOfConcurrentTransactions - 1000000
Default value: 8192
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfConcurrentIndexOperations**
Description: Maximum number of index operations that can be concurrently active on all DB nodes.

Unit: Index operations

Value range: 0 - 1000000
Default value: 8192
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfFiredTriggers**
Description: Maximum number of concurrently fired triggers that can be handled in all concurrent transactions. This currently equals how many operations that modify indexes that can be maintained concurrently.

Unit: Fired triggers

Value range: 0 - 1000000
Default value: 1000
Can be updated: Yes, after system restart

Parameter: [DB] **MaxNoOfConcurrentTransactions**
Description: Maximum number of transactions performed concurrently on the DB nodes.

Unit: Transactions
Value range: ≥ 1
Default value: 4096
Can be updated: Yes, after system restart but not at node restart

Parameter: [DB] **NoOfIndexPages**
Description: Size of memory allocated for primary key indices.
Unit: 8 KB pages
Value range: Minimum 1, Maximum is OS dependent (Solaris 4GB, Linux 3GB, Windows2000 3GB)
Default value: 3000
Can be updated: Yes, after node and system restart. Parameter should not be decreased.

Parameter: [DB] **NoOfDataPages**
Description: Size of memory allocated for database records
Unit: 8 KB pages
Value range: Minimum 1, Maximum is OS dependent (Solaris 4GB, Linux 3GB, Windows2000 3GB)
Default value: 10000
Can be updated: Yes, after node and system restart. Parameter should not be decreased.

Parameter: [DB] **TimeToWaitAlive**
Description: Sets first time for DB and API nodes to wait for all other nodes to start during system restart.
Unit: Seconds
Value range: 1-32
Default value: 25
Can be updated: Yes

Parameter: [DB] **HeartBeatIntervalDbDb**
Description: Heartbeat interval between DB nodes. A node that misses three heartbeats is assumed crashed.
Unit: Milliseconds
Value range: ≥ 10
Default value: 1500
Can be updated: Yes, should be done gradually node-by-node

Parameter: [DB] **HeartBeatIntervalDbApi**
Description: Heartbeat interval between DB and API nodes. Three missed heartbeats lead to closing down the API connection.
Unit: Milliseconds
Value range: ≥ 100
Default value: 1500
Can be updated: Yes, should be done gradually node-by-node

Parameter: [DB] **TimeBetweenInactiveTransactionAbortCheck**
Description: Specifies how often a transaction is checked for inactivity.
Unit: Milliseconds
Value range: ≥ 1000
Default value: 1000
Can be updated: Yes

Parameter: [DB] **TransactionInactiveTimeBeforeAbort**
Description: Specifies how long time a transaction is allowed to remain inactive before it is aborted.
Unit: Milliseconds
Value range: ≤ 10000
Default value: 1000
Can be updated: Yes

Parameter: [DB] **TimeBetweenLocalCheckpoints**
Description: Sets total volume of operations that are executed between local checkpoints (LCP), i.e. how much log space must be used before the next LCP is performed. For instance, if this parameter is set to 20, then a new local checkpoint is started whenever the log in main memory contains 2^{20} (32-bit) words. However, every second also counts as 32 words regardless of whether or not any operations are actually executed. Setting this parameter to zero will result in constantly running local checkpoints, thus minimizing the size of the log in the main memory.
Unit: Logarithmic scale
Value range: 0 to 31
Default value: 20
Can be updated: Yes

Parameter: [DB] **TimeBetweenGlobalCheckpoints**
Description: Sets time interval between global checkpoints (GCP).
Unit: Milliseconds
Value range: 10 – 32000
Default value: 2000
Can be updated: Yes

Parameter: [DB] **NoOfFragmentLogFiles**
Description: Sets the number of fragment log files (or REDO logs) in each set of log files. There are four sets of log files, each containing one or more 16 MB log files. The log record of any operation carries an overhead of 68 bytes plus the size of the primary key involved and 4 bytes per each attribute that is part of a write operation. Hence, updating 100 bytes of data in 25 32-bit attributes with a 32-bit key uses 272 bytes of log space.

Unit: Set of log files
Value range: ≥ 1
Default value: 8
Can be updated: No

Parameter: [DB] **NoOfDiskPagesToDiskDuringRestartTUP**
Description: Specifies how many pages will be sent to disk per 100 milliseconds during a system restart.
Unit: Disk page per 100 milliseconds
Value range: ≥ 1
Default value: 50
Can be updated: Yes

Parameter: [DB] **NoOfDiskPagesToDiskAfterRestartTUP**
Description: Specifies number of pages sent to disk per 100 milliseconds after a system restart.
Unit: Disk page per 100 milliseconds
Value range: ≥ 1
Default value: 10
Can be updated: Yes

Parameter: [DB] **NoOfDiskPagesToDiskDuringRestartACC**
Description: Specifies number of pages sent to disk per 100 milliseconds during system restart.
Unit: Disk pages per 100 milliseconds
Value range: ≥ 1
Default value: 50
Can be updated: Yes

Parameter: [DB] **NoOfDiskPagesToDiskAfterRestartACC**
Description: Specifies number of pages sent to disk per 100 milliseconds after system restart.
Unit: Disk pages per 100 milliseconds
Value range: ≥ 1
Default value: 10
Can be updated: Yes

Parameter: [DB] **ArbitrationTimeout**
Description: A database partition waits this long for signal from arbitrator before crashing.
Unit: Milliseconds
Value range: Integer >= 0
Default value: 1000
Can be updated: Yes (after restart)

Parameter: [DB] **FileSystemPath**
Description: Location of database node data.
Unit: N/A
Value range: Valid directory name (directory must exist)
Default value: None
Can be updated: Yes (after restart)

[TCP] Parameters for TCP connections (transporters)

Parameter: [TCP] **NodeId1**
Description: Identifies the first node for two-way communication.
Unit: N/A
Value range: Valid Node Id
Default value: None
Can be updated: No

Parameter: [TCP] **NodeId2**
Description: Identifies the second node for two-way communication.
Unit: N/A
Value range: Valid Node Id
Default value: None
Can be updated: No

Parameter: [TCP] **IpAddress1 (optional)**
Description: Specifies the IP address of the first node. (This parameter is used to map the communication to an IP address instead of a hostname when the machine hosting the process has more than one network interface and is part of two or more subnets).
Unit: N/A
Value range: Valid IP address
Default value: None
Can be updated: Yes, after system restart.

Parameter: [TCP] **IpAddress2 (optional)**
Description: Specifies the IP address of the second node. See **IpAddress1** above.
Unit: N/A
Value range: Valid IP address
Default value: None
Can be updated: Yes, after system restart.

Parameter: [TCP] **PortNumber**
Description: Port number that the communicating nodes identified by NodeId1 and NodeId2 will use.
Unit: N/A
Value range: Valid port number
Default value: None

Can be updated: Yes, after system restart.

Parameter: [TCP] **SendBufferSize**
Description: Specifies the number of signals that may be queued for sending. One item consists of one 120-byte signal.
Unit: item
Value range: ≥ 1
Default value: 2000
Can be updated: Yes, after system restart.

Parameter: [TCP] **MaxReceiveSize**
Description: Specifies the number of items that will be read from the port at one time. Each signal contains 120 bytes.
Unit: Signals
Value range: ≥ 1
Default value: 512
Can be updated: No

Parameter: [TCP] **SendSignalId**
Description: Specifies whether a signal ID is sent in each signal. Only used for debugging.
Unit: N/A
Value range: Y/N
Default value: Y
Can be updated: No

Parameter: [TCP] **Compression**
Description: Enables compression for TCP connection. Not yet implemented.
Unit: N/A
Value range: Y/N
Default value: N
Can be updated: No

Parameter: [TCP] **Checksum**
Description: Specifies whether or not checksum check is performed.
Unit: N/A
Value range: Y/N
Default value: Y
Can be updated: No

[SCI] Parameters for SCI connections (transporters)

Most parameters for TCP can be used for SCI as well. In addition we have the following parameters:

Parameter: [SCI] **SharedBufferSize**
Description: Specifies the size of the SCI mapped memory segment.

Unit: Bytes
Value range: >256000
Default value: 1048576
Can be updated: Yes, after system restart

Parameter: [SCI] **SendLimit**
Description: Specifies the size of the SCI mapped memory segment.
Unit: Bytes
Value range: 512 to 16184
Default value: 2048
Can be updated: Yes, after system restart

[DB], [API], [MGM] Parameters for database, application and management nodes

Parameter: [DB] **Id,** [API] **Id,** [MGM] **Id**
Description: Identifier of node in the cluster (DB, API, or MGM).
Unit: N/A
Value range: 1 to verified number (DB: 48, API: 64, MGM: no limit)
Default value: None
Can be updated: No

Parameter: [DB] **ExecuteOnComputer,** [API] **ExecuteOnComputer,** [MGM] **ExecuteOnComputer**
Description: Specifies **Id** of the computer that hosts the node.
Unit: N/A
Value range: Valid computer **Id**
Default value: None
Can be updated: No

[API], [MGM] Parameters for application and management nodes

Parameter: [API] **ArbitrationRank,** [MGM] **ArbitrationRank**
Description: If value is > 0, a process on this node can be asked to arbitrate in case of network partitioning. The kernel looks for arbitrators in rank order 1, 2. If no arbitrator is configured or running, there must be (strictly) more than half of the NDB nodes alive for the database to continue running.
Unit: N/A
Value range: Integer 0, 1, 2
Default value: 2 (Is arbitrator)
Can be updated: Yes, after system restart

Parameter: [API] **ArbitrationDelay,** [MGM] **ArbitrationDelay**

Description: Arbitrator waits this long for requests. If set to zero, then first request wins immediately.
Unit: Milliseconds
Value range: Integer >= 0
Default value: 0
Can be updated: Yes, after system restart

[MGM] Parameters for management nodes

Parameter: [MGM] **LogDestination**
Description: String describing zero or more log destinations. The logging system supports logging to CONSOLE, SYSLOG and FILE. Log destinations are separated by a semi-colon ';'. Arguments to a log destination are separated by a comma ','.
1. The CONSOLE destination takes no arguments.
Example: CONSOLE

2. The SYSLOG destination takes one argument; facility, the syslog facility to use. Valid values for facility are: auth, authpriv, cron, daemon, ftp, kern, lpr, mail, news, syslog, user, uucp, local0, local1, local2, local3, local4, local5, local6, local7. (Note that some facilities may be unsupported on platforms where they are not available.)
Example: SYSLOG:facility=local0

3. The FILE destination takes three arguments:
filename Which file to send log data to (mandatory)
maxsize Maximum file size before the files are rolled.

This number can be ended with "k" or "g" for

kilobytes or gigabytes.
maxfiles Maximum number of rolled files.
Example: FILE:filename=cluster.log,maxsize=1000000,maxfiles=6

Multiple log destinations can be given as in the following example:
CONSOLE;SYSLOG:facility=local0;FILE:filename=/var/log/mgm
Unit: N/A
Value range: All valid strings
Default value:
FILE:filename=cluster.log,maxsize=1000000,maxfiles=6
Can be updated: Yes, after system restart

3.6. Configuration of DB and API nodes

DB and API nodes get information about their node id and the location of the management server through their local configuration files (Ndb.cfg). The directives placed in these files are identical for both node types and is of the following form:

`OwnProcessId` <*NodeId*>
<*MGMHostName*> <*MGMPortNumber*>

Where:

- <NodeId> is the node identity of the node.
- <MGMHostname> is the hostname or the IP-address of the management server.
- <MGMPortNumber> is the port number of the management server.

Note that no colons are used to separate parameter names and values.

Below is an example of a local configuration file:

```
# Sample local configuration file
OwnProcessId 5
mycomputer62  25006
```

4. Management Commands

In addition to the configuration file (config.ini), the management servers are also controlled through a command line interface. The command line interface is available in the same terminal window as the started management server or through a separate management client process. This interface is the main administrative interface to NDB Cluster.

The management server has the following basic commands. Below, <Id> denotes either a database node id (e.g. 21) or the keyword **all** that indicates that the command should be applied to all database nodes in the NDB Cluster.

- **help**
  Prints information on all available commands.
- **show**
  Prints information on the status of the cluster.
- **show config**
  Prints current configuration.
- **show parameters**
  Prints information about all configuration parameters.
- <*Id*> **start**
  Start a database node identified with *Id* or all database nodes.
- <*Id*> **stop**
  Stop a database node identified with *Id* or all database nodes.
- <*Id*> **restart [-n] [-i]**
  Restart a database node identified with *Id* or all database nodes.
- <*Id*> **status**
  Displays status information for database node identified with *Id* (or all database nodes).
- **enter single user mode** <*nodeid*>
  Enters single user mode where only the API with node id <*nodeid*> is allowed to access the database system
- **exit single user mode**
  Exists single user mode allowing all APIs to access the database system
- **quit**
  Terminates management server or management client.

Commands for the event logs are given in the next section and commands for backup and restore are given in the separate chapter on Backup and Restore.

4.1. Event Logs

NDB Cluster has two event logs, the *cluster log* and the *node log*. The cluster log is a log of the whole NDB Cluster and this log can have multiple destinations (file, management server console window or syslog). The node log is a log that is local to each database node and is written to the console window of the database node. The two logs can be set to log different subsets of the list of events.

Note: The cluster log is the recommended log. The node log is only intended to be used during application development or for debugging application code.

Each reportable event has the following properties:

- **Category** (STARTUP, SHUTDOWN, STATISTICS, CHECKPOINT, NODERESTART, CONNECTION, ERROR, INFO)
- **Priority** (1-15 where 1 – Most important, 15 – Least important)
- **Severity** (ALERT, CRITICAL, ERROR, WARNING, INFO, DEBUG)

The two logs (the cluster log and the node log) can be filtered on these properties.

Cluster Log

The following commands are related to the cluster log:

- **clusterlog on**
  Turn cluster log on.
- **clusterlog off**
  Turn cluster log off.
- **clusterlog info**
  Information about cluster log settings.
- *<Id>* **clusterlog** *<category>=<threshold>*
  Log *category* events with priority less than or equal to *threshold* in the cluster log.
- **clusterlog filter** *<severity>*
  Toggles cluster logging of the specified severity type on/off.

The following table describes the default setting (for all database nodes) of the cluster log category threshold. If an event has a priority with a value lower than or equal to the priority threshold, then it is reported in the cluster log. Note that the events are reported per database node and that the thresholds can be set differently on different nodes.

Category	Default threshold (All database nodes)
STARTUP	7
SHUTDOWN	7
STATISTICS	7
CHECKPOINT	7
NODERESTART	7
CONNECTION	7
ERROR	15
INFO	7

The threshold is used to filter events within each category. For example: A STARTUP event with a priority of 3 is never sent unless the threshold for STARTUP is changed to 3 or lower. Only events with priority 3 or lower are sent if the threshold is 3.

The event severities are (corresponds to UNIX syslog levels):

1. **ALERT**
   A condition that should be corrected immediately, such as a corrupted system database
2. **CRITICAL**
   Critical conditions, such as device errors or out of resources.
3. **ERROR**
   Conditions that should be corrected, such as configuration errors.
4. **WARNING**
   Conditions that are not error conditions but might require handling
5. **INFO**
   Informational messages
6. **DEBUG**
   Messages used during development of NDB Cluster

Syslog's **LOG_EMERG** and **LOG_NOTICE** are not used/mapped.

The event severities can be turned on or off. If the severity is *on* then all events with priority less than or equal to the category thresholds are logged. If the severity is *off* then no events belonging to the severity are logged.

Node log

The following commands are related to the node log:

- <*Id*> **loglevel** <*levelnumber*>
  Set logging level for database process with *Id* to the value of *levelnumber*

4.2. Log events

All reportable events are listed below.

Event	Category	Priority	Severity	Description
DB nodes connected	CONNECTION	8	INFO	
DB nodes disconnected	CONNECTION	8	INFO	
Communication closed	CONNECTION	8	INFO	API & DB nodes connection closed
Communication opened	CONNECTION	8	INFO	API & DB nodes connection opened
Global checkpoint started	CHECKPOINT	9	INFO	Start of a GCP, i.e., REDO log is written to disk
Global checkpoint completed	CHECKPOINT	10	INFO	GCP finished
Local checkpoint started	CHECKPOINT	7	INFO	Start of local check pointing, i.e., data is written to disk. LCP Id and GCI Ids (keep and oldest restorable)
Local checkpoint completed	CHECKPOINT	8	INFO	LCP finished
LCP stopped in calc keep GCI	CHECKPOINT	0	ALERT	LCP stopped!
Local checkpoint fragment completed	CHECKPOINT	11	INFO	A LCP on a fragment has been completed
Report undo log blocked	CHECKPOINT	7	INFO	Reports undo logging blocked due buffer near to overflow
DB node start phases initiated	STARTUP	1	INFO	NDB Cluster starting
DB node all start phases completed	STARTUP	1	INFO	NDB Cluster started
Internal start signal received STTORRY	STARTUP	15	INFO	Internal start signal to blocks received after restart finished
DB node start phase X completed	STARTUP	4	INFO	A start phase has completed
Node has been successfully included into the cluster	STARTUP	3	INFO	President node, own node and dynamic id is shown
Node has been refused to be included into the cluster	STARTUP	8	INFO	
DB node neighbours	STARTUP	8	INFO	Show left and right DB nodes neighbours
DB node shutdown initiated	STARTUP	1	INFO	
DB node shutdown aborted	STARTUP	1	INFO	
New REDO log started	STARTUP	10	INFO	GCI keep X, newest restorable GCI Y
New log started	STARTUP	10	INFO	Log part X, start MB Y, stop MB Z
Undo records executed	STARTUP	15	INFO	

Completed copying of dictionary information	NODERESTART	8	INFO	
Completed copying distribution information	NODERESTART	8	INFO	
Starting to copy fragments	NODERESTART	8	INFO	
Completed copying a fragment	NODERESTART	10	INFO	
Completed copying all fragments	NODERESTART	8	INFO	
Node failure phase completed	NODERESTART	8	ALERT	Reports node failure phases
Node has failed, node state was X	NODERESTART	8	ALERT	Reports that a node has failed
Report whether an arbitrator is found or not	NODERESTART	6	INFO	7 different cases - President restarts arbitration thread [state=X] - Prepare arbitrator node X [ticket=Y] - Receive arbitrator node X [ticket=Y] - Started arbitrator node X [ticket=Y] - Lost arbitrator node X – process failure [state=Y] - Lost arbitrator node X – process exit [state=Y] - Lost arbitrator node X <error msg>[state=Y]
Report arbitrator results	NODERESTART	2	ALERT	8 different results - Arbitration check lost – less than 1/2 nodes left - Arbitration check won – node group majority - Arbitration check lost – missing node group - Network partitioning – arbitration required - Arbitration won – positive reply from node X - Arbitration lost – negative reply from node X - Network partitioning – no arbitrator available - Network partitioning – no arbitrator configured
GCP take over started	NODERESTART	7	INFO	
GCP take over completed	NODERESTART	7	INFO	
LCP take over started	NODERESTART	7	INFO	
LCP take completed (state = X)	NODERESTART	7	INFO	
Report transaction statistics	STATISTICS	8	INFO	# of: transactions, commits, reads, simple reads, writes, concurrent operations, attribute info, aborts
Report operations	STATISTICS	8	INFO	# of operations
Report table create	STATISTICS	7	INFO	
Report job scheduling statistics	STATISTICS	9	INFO	Mean Internal job scheduling statistics
Sent # of bytes	STATISTICS	9	INFO	Mean # of bytes sent to node X

Received # of bytes	STATISTICS	9	INFO	Mean # of bytes received from node X
Memory usage	STATISTICS	5	INFO	Data and Index memory usage (80%, 90% and 100%)
Transporter errors	ERROR	2	ERROR	
Transporter warnings	ERROR	8	WARNING	
Missed heartbeats	ERROR	8	WARNING	Node X missed heartbeat # Y
Dead due to missed heartbeat	ERROR	8	ALERT	Node X declared dead due to missed heartbeat
General warning events	ERROR	2	WARNING	
Sent heartbeat	INFO	12	INFO	Heartbeat sent to node X
Create log bytes	INFO	11	INFO	Log part, log file, MB
General info events	INFO	2	INFO	

An event report has the following format in the logs:

```
<date & time in GMT> [<any string>] <event severity> -- <log message>

  09:19:30 2003-04-24 [NDB] INFO -- Node 4 Start phase 4 completed
```

4.3. Singe user mode

Enter single user mode

Single user mode allows the database administrator to restrict access to the database system to only one application (API node). When entering single user mode all connections to all APIs will be gracefully closed and no transactions are allowed to be started. All running transactions are aborted.

When the NDB Cluster has entered single user mode (use the **all status** command to see when the state has entered the single user mode), only the allowed API node is granted access to the database.

Example:

enter single user mode 5

After executing this command and after NDB Cluster has entered the single user mode, the API node with node id 5 becomes the single user of the NDB Cluster.

The node specified in the command above must be of API type. Any attempt to specify any other type of node will be rejected.

Note: If the API node with node id 5 is running when executing **enter single user mode 5**, all transactions running on API node 5 will be aborted, the connection to the API is closed, and the API must be restarted.

Exit single user mode

The command **exit single user mode** alters the state of the NDB Cluster DB nodes from "single user mode" to "started".

APIs waiting for a connection, i.e. for NDB Cluster to become ready, are now allowed to connect. The API denoted as the single user continues to run, if it is connected, during and after the state transition.

Example:

exit single user mode

Single user mode and node failures

Best practice in case of node failures when running in single user mode is to:

1. Finish all single user mode transactions
2. Issue the command **exit single user mode**
3. Restart database nodes

Or restart database nodes prior to entering single user mode.

5. Backup and Restore

This chapter describes how to create a backup and later restore the backup to a database.

5.1. Concepts

A backup is a snapshot of the database at a given time. The backup contains three main parts:

1. Meta data (what tables exists etc)
2. Table records (data in tables)
3. A log of committed transactions

Each of these parts is saved on all nodes participating in a backup.

During backup each node saves these three parts to disk into three files:

1. **BACKUP-<*BackupId*>.<*NodeId*>.ctl**
   The control file which contain control information and meta data.
2. **BACKUP-<*BackupId*>-0.<*NodeId*>.data**
   The data file that contain the table records.
3. **BACKUP-<*BackupId*>.<*NodeId*>.log**
   The log file that contain the committed transactions.

Above <*BackupId*> is an identifier for the backup and <*NodeId*> is the node id of the node creating the file.

Meta data

The meta data consists of table definitions. All nodes have the exact same table definitions saved on disk.

Table records

The table records are saved per fragment. Each fragment contains a header that describes which table the records belong to. After the list of records there is a footer that contains a checksum for the records.
Different nodes save different fragments during the backup.

Committed log

The committed log contains committed transaction made during the backup. Only transactions on tables stored in the backup are stored in the log. The different nodes in the backup saves different log records as they host different database fragments.

5.2. Using the management server to create a backup

Before starting make sure that the NDB Cluster is properly configured for backups.

1. Start management server.
2. Execute the command "start backup".
3. The management server will reply with a message "Start of backup ordered".
   This means that the management server has submitted the request to NDB Cluster, but has not yet received any response.
4. The management server will reply "Backup <*BackupId*> started" where <*BackupId*> is the unique identifier for this particular backup. This will also be saved in the cluster log (if not configured otherwise).
   This means that NDB Cluster has received and processed the backup request. It does not mean that the backup has completed.
5. The management server will when the backup is finished reply "Backup <*BackupId*> completed".


```
NDB Cluster Management Server started on port 37123.
NDB Cluster Statistics available on port 2199.
NDB>
NDB> start backup
Start of backup ordered
NDB>
Backup 12 started

Backup 12 completed
 StartGCP: 6400 StopGCP: 6400
 #Records: 4096 #LogRecords: 0
 Data: 66176 bytes Log: 0 bytes

NDB>
```

Note that if there is not any backup with id *BackupId* running when it is aborted, the management server will not reply anything. However there will be a line in the cluster.log mentioning that an "invalid" abort command has been filed.

```
NDB Cluster Management Server started on port 37123.
NDB Cluster Statistics available on port 2199.
NDB>
NDB> abort backup 12
Abort of backup 12 ordered
NDB>
Backup 12 has been aborted reason 123
```

tables.

The restore program needs to be started in a directory containing an Ndb.cfg file. The backup files must be present in the same directory.

The backup can be restored to a database with a different configuration than it was created from.

For example, consider if a backup (with id 12) created in an NDB Cluster with two database nodes (with node id 2 and node id 3) that should be restored to an NDB Cluster with four nodes. The restore program then has to be executed two times (one for each database node in the NDB Cluster where the backup was taken) as described in the box below.

```
BACKUP-12> restore -backupid 12 -nodeid 2 -restore -restore_meta
Connected to ndb!!
Created table SERVER
Created table SESSION
Created table GROUP
Created table SUBSCRIBER
Restored 152100 tuples and 123 log entries

BACKUP-12> restore -backupid 12 -nodeid 3 -restore
Connected to ndb!!
Restored 13423 tuples and 12312 log entries
```

and the log buffer.

5.6. Troubleshooting

If an error code is returned when issuing a backup request, then check that there is enough memory allocated for the backup (i.e. the configuration parameters). Also check that there is enough space on the hard drive partition of the backup target.

6. Using Scalable Coherent Interface (SCI)

NDB Cluster supports the SCI (IEEE std. 1596-1992) interconnect. In order to use SCI with NDB Cluster, the following hardware and software are required:

Hardware

Two Dolphin ICS D330 SCI adapter cards in every computer.
Two Dolphin ICS D535 SCI switches (if there are more than two computers to interconnect)

Software

Dolphin ICS SCI driver version 1.11.15 installed on every computer.

6.1. Example configuration file

```
##############################################################################
#  System configuration file for NDB Cluster Version 2.00
#
#  MySQL AB
#  Web: www.mysql.com
##############################################################################

[DB DEFAULT]
NoOfReplicas: 2

# SciId0 and SciId1 corresponds to SCI node identities of the SCI adapters.
# SciId0 maps to the SCI node id of adapter 0 (adapter id) and
# SciId1 maps to the SCI node id of adapter 1.
# The SCI node id is configurable.
#
# E.g., in this configuration example the computer with id 1
# has been assigned the SCI node 324 and 328.
#
# NOTE: All SCI adapters with adapter id 0 must be connected to one switch and
# all SCI adapters with adapter id 1 must be connected to the other switch.

[COMPUTER DEFAULT]
ByteOrder: Big

[COMPUTER]
Id: 1
HostName: 10.0.1.1
SciId0: 324
SciId1: 328

[COMPUTER]
Id: 2
ByteOrder: Big
HostName: 10.0.2.1
SciId0: 260
SciId1: 264

[COMPUTER]
Id: 3
ByteOrder: Big
HostName: 10.0.40.1
SciId0: 388
SciId1: 392

[MGM]
Id: 1
ExecuteOnComputer: 1
```

```
ArbitrationRank: 1

[DB]
Id: 2
ExecuteOnComputer: 1

[DB]
Id: 3
ExecuteOnComputer: 2

[API]
Id: 4
ExecuteOnComputer: 3
ArbitrationRank: 2

[API]
Id: 5
ExecuteOnComputer: 3
ArbitrationRank: 2

# The following configuration has TCP connections between the DB nodes
# and the Management server. SCI is used between the DB nodes and
# the API nodes.  SharedBufferSize should not be lower than 200000.

[TCP]
NodeId1: 1
NodeId2: 2
PortNumber: 37125

[TCP]
NodeId1: 1
NodeId2: 3
PortNumber: 37126

[SCI]
NodeId1: 2
NodeId2: 3
SharedBufferSize: 200000
SendLimit: 2048

[SCI]
NodeId1: 2
NodeId2: 4
SharedBufferSize: 200000
SendLimit: 2048

[SCI]
NodeId1: 2
NodeId2: 5
SharedBufferSize: 200000
SendLimit: 2048

[SCI]
NodeId1: 3
NodeId2: 4
SharedBufferSize: 200000
SendLimit: 2048

[SCI]
NodeId1: 3
NodeId2: 5
SharedBufferSize: 200000
SendLimit: 2048
```

7. NDB SQL

Please note that this chapter is in high state of flux, due to the integration with the MySQL server. NDB features currently incomplete or planned for a future release are marked with "**Future**". Some of the mentioned features may already be supported directly through MySQL.

7.1. SQL

Data types

type	description
CHAR(n)	Fixed-width blank-padded string
VARCHAR(n)	Variable length string
BINARY(n) VARBINARY(n)	Binary strings
SMALLINT	Integer 16 bits
INT INTEGER	Integer 32 bits
BIGINT	Integer 64 bits
DECIMAL(m,n)	Exact number with precision and scale **Future**
REAL	Float 32 bits
FLOAT DOUBLE PRECISION	Float, at least 64 bits
DATE	Date with precision 1 second **Future**
DATETIME	Date with precision 1 nanosecond (SQL_TYPE_TIMESTAMP)

Integer types may be qualified as UNSIGNED.

Strings and numbers are not currently converted to each other automatically. Following is an error (unlike in Oracle).

```
select 123 + '456' from tab
```

Expressions

Syntax	description
NULL	Null value
12.34e5	Integer or decimal or float constant
'abc'	String constant
+ - * / ( )	Arithmetic operations
||	String concatenation **Future**

Integer and decimal arithmetic is done in BIGINT.
Floating arithmetic is done in DOUBLE PRECISION.
Numeric literals use largest applicable type.
String operations are done in CHAR or in VARCHAR (if any operand is VARCHAR).
String literals are of type CHAR.

Functions: non-aggregate

Syntax	description
SUBSTR LEFT RIGHT	Substring
TO_NUMBER TO_CHAR	Basic conversions **Future**
ROWNUM	Row number in query
SYSDATE	Current date as DATETIME

Functions : aggregate

Syntax	description
COUNT	Count rows or non-NULL values
MIN MAX	Min and max of strings and numbers
SUM AVG	Sum and average of numbers

GROUP BY and HAVING are fully supported.

Predicates

Syntax	description
IS NULL / IS NOT NULL	Test if value is null
< <= = != > >=	Comparisons
LIKE / NOT LIKE	String matching
AND OR NOT	Boolean operators

Create table

An NDB Cluster table requires a primary key. There are two ways to specify it.

Case 1

```
create table t (
    a integer not null,
    b char(20) not null,
    c float,
    primary key(a, b)
)
```

Case 2

A column can be specified as AUTO_INCREMENT. The column has following requirements:

1. it must be the primary key (not just part of one)
2. its type must be one of the integer types

```
create table t (
    a int auto_increment primary key,
    b char(20) not null,
    c float
)
```

The values of an AUTO_INCREMENT column are unique (until wrap-around) and form an ascending sequence. Starting value is 1. Gaps in the sequence are possible.

Default values

Columns can be specified with DEFAULT value which is used on insert if the column is not on the insert list.

```
create table t (
    a int primary key,
    b int default 100
)
insert into t(a) values(1) -- inserts (1,100)
```

The value must evaluate to constant. Using SYSDATE (if allowed at all) evaluates to table creation time.

Logging / nologging

By default tables are created in logging mode, meaning that data is preserved across database restart. The mode can be specified explicitly:

```
create table t1 (. . .) logging
create table t1 (. . .) nologging
```

Schemas

Schemas do not exist in current NDB Cluster. As a convenience, a single period is allowed in table names:

```
create table mydb.mytable (a int primary key)
```

Drop table

Deletes a table, all of its indexes, and all data:

```
drop table t
```

Create and Drop Index

Only unique non-ordered indexes exist currently. Index columns must be not nullable and are stored in same order as underlying table columns.

Examples:

create unique hash index x1 on t1 (b, d)

drop index x1

Select

Features:

- Expressions and predicates
```
select a + b * c
from t
where a ≤ b + c and (b > c or c > 10)
```
- JOIN to any depth
```
select a.x, b.y, c.z
from t1 a, t2 b, t2 c
where a.x + b.y < c.z
```
- ORDER BY
```
select *
from t1, t2
where a1 > 5
order by b1 + b2, c1 desc
```
- DISTINCT
- select distinct * from t
- Aggregates without grouping
```
select count(*), max(a), 1 + sum(b) + avg(c * d)
from t
```
- ```
  Aggregates with grouping
  select a, sum(b)
  from t
  ```
  group by a having sum(c) > 0
  order by a, sum(d)

Limiting number of rows in output

The following two examples produce the same result (limits the output to 10 rows):

- select … where rownum <= 10
- select … limit 10

The following two examples limits the output to 10 rows starting with output row number 20:

- select … limit 20, 10
- select … limit 10 offset 20

(The syntax is as in MySQL and PostgreSQL.)

Major omissions:

- no OUTER JOIN
- no subqueries and no EXISTS clause

Queries are optimized to minimize scans, by using primary keys and existing unique hash indexes. Simple predicates in scans (column compared to constant) are passed to an interpreter in NDB kernel. Joins are done via *nested loops* only.
  ```

- SCAN
  `select * from t where a < b`
- INTERPRETABLE SCAN (faster)
  select * from t1, t2 where t1.a < 10 and t2.b > t1.c + 1
- PRIMARY KEY lookup
  `select * from t where pk = 5 and b > 10`
- NESTED LOOPS / SCAN and PRIMARY KEY lookup
  `select * from t1, t2, t3 where t1.pk = t2.x and t2.pk = t3.y`

Insert and write

Both VALUES and sub-query variants can be used.

```
insert into t(a, c) values (123, 'abc')
insert into t1(a, c) select a + 10 * b, c from t2
```

For convenience, the non-standard *MySQL* syntax is also supported.

```
insert into t set a = 123, c = 'abc'
```

The non-standard operation WRITE is used exactly like INSERT. The record is updated if it exists. Otherwise a new record is inserted.

```
write into t(a, c) values (123, 'abc')
```

Update

Update allows no subqueries. Update is optimized to minimize scans and reads.

- SCAN
  `update t set a = b + 5, c = d where c > 10`
- PRIMARY KEY lookup
  `update t set a = b + 5, c = d where pk = 5 and c > 10`
- PRIMARY KEY direct
  `update t set a = 5, c = 7 where pk = 5`

Delete

Delete allows no subqueries. Delete is optimized to minimize scans and reads.

- SCAN
  `delete from t where c > 10`
- PRIMARY KEY lookup
  `delete from t where pk = 5 and c > 10`
- PRIMARY KEY direct
  `delete from t where pk = 5`

7.2. Data formats

SQL types are represented as NDB Cluster types as follows.

SQL type	NDB type
CHAR(n)	String(n), blank-padded to n

VARCHAR(n)	String(n+2), zero-padded to n, length in last 2 bytes (big-endian)
Integers	Signed(x) or UnSigned(x), x=16,32,64, native format
Floats	Float(x), x=32,64, native format
DATETIME	String(12) = cc yy mm dd HH MM SS \0 ff ff ff ff (big-endian)

NDB Cluster limitations

- Isolation level is READ COMMITTED. A scan (non-primary-key select of several rows) does not see consistent data.
- Inserting into a table from itself is likely to cause a deadlock or a random result.
  no: `insert into t(a, b) select a*100, b+100 from t`
- Number of uncommitted rows is limited by an NDB Cluster configuration parameter **`MaxNoOfConcurrentOperations`** (typical default 4096). To delete all rows from a large table one may need to do repeatedly:
  `delete from t where rownum < 4000`

Known problems NDB Cluster Release 2.10

Following lists specific known problems.

- ORDER BY works only with expressions, not with column aliases or positions.
  no: `select a+b x from t order by x`
  no: `select * from t order by 1, 2, 3`
- Join optimizer does not always minimize number of scans. Changing the order of tables in the statement may help.
- Indexes for a table must be dropped before dropping the table.

8. Troubleshooting

8.1. The management server (mgmtsrvr) does not start

- Check that the configuration file (config.ini) has correct values. Try with one of the demo versions of the config.ini to check if that works.

8.2. A database node (ndb) fails to connect to a management server (mgmtsrvr)

1. Check that the management server is started on the correct computer as specified in the config.ini file
2. Check that you can ping all the machines of your cluster, by name and by IP address.
    a. If you can not ping by name, then check your /etc/hosts file or DNS
    b. If you can not ping by IP address, then check your network connections
3. Check that the management server is started using the correct port number.
4. If any database node starts but reports "Cannot find Ndb.cfg", then make sure that you are starting the node in the directory that contains the correct Ndb.cfg file.
5. Check in config.ini that the database node is configured on the correct computer in the config.ini (see parameter **ExecuteOnComputer**).
6. If a database node starts but reports: "Warning: Config data could not be retrieved from any management server…", then:
    a. Check the port number in the Ndb.cfg file. Is it the same port number as the port used by the management server?
    b. Check the hostname in the Ndb.cfg file. Is it the same host as the management server is executing on?
7. If the database node starts but reports: "The nodeid is not the actual node id".
    a. Check that the OwnProcessId in Ndb.cfg corresponds to the id of the process in the config.ini file and that the database node is configured to be executing on the correct computer (see parameter **ExecuteOnComputer**).

8.3. Database node (ndb) does not start

- Make sure that the management server is up and running.
- Check the local configuration file (Ndb.cfg). This file must be in the working directory of the ndb process.
- Check that the local configuration file (Ndb.cfg) contains the correct location (IP address and port number) of the management server.
- Check that there is enough free space on the disk for the database node. The file system disk directory for each database node is normally specified in the configuration file (config.ini) stored in the working directory of the management server.

8.4. Database node (ndb) does not start because of NdbMem_Allocate error

- Check that the memory requirements are fulfilled. On Windows at least 256 Mbyte RAM is needed. A typical one-node configuration requires 132 Mbyte for the database node and a typical two-node configuration requires 92 Mbyte per database node.
- If you cannot satisfy the memory requirements (if you are running Windows, you may check these using the Windows Task Manager), you may reduce the memory allocated for data and indexes (i.e. reduce the **MemorySpaceTuples** and the **MemorySpaceIndexes** configuration parameters in the configuration file config.ini).

8.5. An application can not be started

- Make sure that the management server is up and running.
- Check the local configuration file (Ndb.cfg). This file must be in the working directory of the application process.
- Check that the local configuration file (Ndb.cfg) contains the correct location (IP address and port number) of the management server.
- Make sure that all database nodes are up and running. This can be done by typing 'all status' in the management server command line.

8.6. Database node fails while database is loaded with information

- Make sure that there is enough main memory for the information. This is set in the configuration file. There are two parameters, one for the actual table data and one for the indexes. See configuration section in this document.
  Note: After changing the configuration file, the management server and the database nodes needs to be re-started.

8.7. One or more database nodes fail during startup

- If you have changed the configuration or the file system has been corrupted, you can try to delete the file system of each database node. Start the database nodes with the –i flag to delete the file system.

8.8. A database node fails during operation

- Check the error log for the database node (error.log). This file is located in the working directory of the database node process. Note that the last error may not be the last error written in the file.
- Check the cluster log (cluster.log). This file is located in the working directory of the management server.

8.9. Compilation problems of application programs

- To compile NDB API programs use Visual Studio Version 7.
- Set "Configuration Name=Release|Win32".
- The following project settings are used internally by the NDB Cluster team at MySQL when we are using Visual Studio:

  On **File** menu click **New project** (Select name and location to store it).
  On **Build** menu click **Configuration Manager**, Set to: Release.
  Set "Application Setting" to "Console application".
  General / Whole Program Optimization
  - Set to: yes (set to "no" if you want to debug your code).

  Debugging / Working Directory
  - Set to appropriate API directory (where you will run your application program).

  C/C++ / General / Addition Include Directories
  - Add: ndb\include; ndb\include\ndbapi; ndb\include\portlib

  C/C++ / Code Generation / Code generation / Runtime Library
  - Set to: Multi-threaded (/MT) (set to "/MTD" if you want to debug).

  C/C++ / Preprocessor / Preprocessor definitions
  - Add: NDB_WIN32

  Linker / General / Additional Library Directories
  - Add: \NDB\lib

  Linker / Input / Additional Dependencies
  - Add: NDB_API.lib ws2_32.lib sisci_api_mt.lib

8.10. After a power failure, one or more database nodes do not restart

- This could happen if the file system was corrupted during the failure. A solution to this problem is to restart the database node(s) with a clean (erased) file system. This can be done by starting the node(s) with the –i parameter (that is start the database node(s) with the “ndb –i” command).

8.11. High load problems (many concurrent data-intensive transactions)

- The transporter buffers used by NDB Cluster has a configurable size. Make sure that **SendBufferSize** and **MaxReceiveSize** (in the configuration file config.ini) are correctly set for all transporters.

8.12. Other problems

- The MySQL mailing lists can be found via http://lists.mysql.com/.
- The MySQL bug system is at http://bugs.mysql.com/.
- MySQL AB also offers commercial support, plese see http://www.mysql.com/support/ for further information.