**16.3 The MEMORY Storage Engine**

The MEMORY storage engine (formerly known as HEAP) creates special-purpose tables with contents that are stored in memory. Because the data is vulnerable to crashes, hardware issues, or power outages, only use these tables as temporary work areas or read-only caches for data pulled from other tables.

**Table 16.4 MEMORY Storage Engine Features**

| **Feature** | **Support** |
| --- | --- |
| **B-tree indexes** | Yes |
| **Backup/point-in-time recovery** (Implemented in the server, rather than in the storage engine.) | Yes |
| **Cluster database support** | No |
| **Clustered indexes** | No |
| **Compressed data** | No |
| **Data caches** | N/A |
| **Encrypted data** | Yes (Implemented in the server via encryption functions.) |
| **Foreign key support** | No |
| **Full-text search indexes** | No |
| **Geospatial data type support** | No |
| **Geospatial indexing support** | No |
| **Hash indexes** | Yes |
| **Index caches** | N/A |
| **Locking granularity** | Table |
| **MVCC** | No |
| **Replication support** (Implemented in the server, rather than in the storage engine.) | Limited (See the discussion later in this section.) |
| **Storage limits** | RAM |
| **T-tree indexes** | No |
| **Transactions** | No |
| **Update statistics for data dictionary** | Yes |

* [When to Use MEMORY or NDB Cluster](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-compared-cluster)
* [Partitioning](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-partitioning)
* [Performance Characteristics](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-performance-characteristics)
* [Characteristics of MEMORY Tables](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-characteristics-of-memory-tables)
* [DDL Operations for MEMORY Tables](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-ddl-operations-for-memory-tables)
* [Indexes](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-indexes)
* [User-Created and Temporary Tables](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-user-created-and-temporary-tables)
* [Loading Data](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-loading-data)
* [MEMORY Tables and Replication](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-tables-replication)
* [Managing Memory Use](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-managing-memory-use)
* [Additional Resources](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#memory-storage-engine-additional-resources)

**When to Use MEMORY or NDB Cluster**

Developers looking to deploy applications that use the MEMORY storage engine for important, highly available, or frequently updated data should consider whether NDB Cluster is a better choice. A typical use case for the MEMORY engine involves these characteristics:

* Operations involving transient, non-critical data such as session management or caching. When the MySQL server halts or restarts, the data in MEMORY tables is lost.
* In-memory storage for fast access and low latency. Data volume can fit entirely in memory without causing the operating system to swap out virtual memory pages.
* A read-only or read-mostly data access pattern (limited updates).

NDB Cluster offers the same features as the MEMORY engine with higher performance levels, and provides additional features not available with MEMORY:

* Row-level locking and multiple-thread operation for low contention between clients.
* Scalability even with statement mixes that include writes.
* Optional disk-backed operation for data durability.
* Shared-nothing architecture and multiple-host operation with no single point of failure, enabling 99.999% availability.
* Automatic data distribution across nodes; application developers need not craft custom sharding or partitioning solutions.
* Support for variable-length data types (including [BLOB](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob) and [TEXT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob)) not supported by MEMORY.

**Partitioning**

MEMORY tables cannot be partitioned.

**Performance Characteristics**

MEMORY performance is constrained by contention resulting from single-thread execution and table lock overhead when processing updates. This limits scalability when load increases, particularly for statement mixes that include writes.

Despite the in-memory processing for MEMORY tables, they are not necessarily faster than [InnoDB](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\innodb-storage-engine.html) tables on a busy server, for general-purpose queries, or under a read/write workload. In particular, the table locking involved with performing updates can slow down concurrent usage of MEMORY tables from multiple sessions.

Depending on the kinds of queries performed on a MEMORY table, you might create indexes as either the default hash data structure (for looking up single values based on a unique key), or a general-purpose B-tree data structure (for all kinds of queries involving equality, inequality, or range operators such as less than or greater than). The following sections illustrate the syntax for creating both kinds of indexes. A common performance issue is using the default hash indexes in workloads where B-tree indexes are more efficient.

**Characteristics of MEMORY Tables**

The MEMORY storage engine does not create any files on disk. The table definition is stored in the MySQL data dictionary.

MEMORY tables have the following characteristics:

* Space for MEMORY tables is allocated in small blocks. Tables use 100% dynamic hashing for inserts. No overflow area or extra key space is needed. No extra space is needed for free lists. Deleted rows are put in a linked list and are reused when you insert new data into the table. MEMORY tables also have none of the problems commonly associated with deletes plus inserts in hashed tables.
* MEMORY tables use a fixed-length row-storage format. Variable-length types such as [VARCHAR](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#char) are stored using a fixed length.
* MEMORY tables cannot contain [BLOB](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob) or [TEXT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob) columns.
* MEMORY includes support for AUTO\_INCREMENT columns.
* Non-TEMPORARY MEMORY tables are shared among all clients, just like any other non-TEMPORARY table.

**DDL Operations for MEMORY Tables**

To create a MEMORY table, specify the clause ENGINE=MEMORY on the [CREATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#create-table) statement.

CREATE TABLE t (i INT) ENGINE = MEMORY;

As indicated by the engine name, MEMORY tables are stored in memory. They use hash indexes by default, which makes them very fast for single-value lookups, and very useful for creating temporary tables. However, when the server shuts down, all rows stored in MEMORY tables are lost. The tables themselves continue to exist because their definitions are stored in the MySQL data dictionary, but they are empty when the server restarts.

This example shows how you might create, use, and remove a MEMORY table:

mysql> **CREATE TABLE test ENGINE=MEMORY**

**SELECT ip,SUM(downloads) AS down**

**FROM log\_table GROUP BY ip;**

mysql> **SELECT COUNT(ip),AVG(down) FROM test;**

mysql> **DROP TABLE test;**

The maximum size of MEMORY tables is limited by the [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) system variable, which has a default value of 16MB. To enforce different size limits for MEMORY tables, change the value of this variable. The value in effect for [CREATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#create-table), or a subsequent [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table) or [TRUNCATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#truncate-table), is the value used for the life of the table. A server restart also sets the maximum size of existing MEMORY tables to the global [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) value. You can set the size for individual tables as described later in this section.

**Indexes**

The MEMORY storage engine supports both HASH and BTREE indexes. You can specify one or the other for a given index by adding a USING clause as shown here:

CREATE TABLE lookup

(id INT, INDEX USING HASH (id))

ENGINE = MEMORY;

CREATE TABLE lookup

(id INT, INDEX USING BTREE (id))

ENGINE = MEMORY;

For general characteristics of B-tree and hash indexes, see [Section 8.3.1, “How MySQL Uses Indexes”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#mysql-indexes).

MEMORY tables can have up to 64 indexes per table, 16 columns per index and a maximum key length of 3072 bytes.

If a MEMORY table hash index has a high degree of key duplication (many index entries containing the same value), updates to the table that affect key values and all deletes are significantly slower. The degree of this slowdown is proportional to the degree of duplication (or, inversely proportional to the index cardinality). You can use a BTREE index to avoid this problem.

MEMORY tables can have nonunique keys. (This is an uncommon feature for implementations of hash indexes.)

Columns that are indexed can contain NULL values.

**User-Created and Temporary Tables**

MEMORY table contents are stored in memory, which is a property that MEMORY tables share with internal temporary tables that the server creates on the fly while processing queries. However, the two types of tables differ in that MEMORY tables are not subject to storage conversion, whereas internal temporary tables are:

* If an internal temporary table becomes too large, the server automatically converts it to on-disk storage, as described in [Section 8.4.4, “Internal Temporary Table Use in MySQL”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#internal-temporary-tables).
* User-created MEMORY tables are never converted to disk tables.

**Loading Data**

To populate a MEMORY table when the MySQL server starts, you can use the [--init-file](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#option_mysqld_init-file) option. For example, you can put statements such as [INSERT INTO ... SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert-select) or [LOAD DATA INFILE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#load-data) into this file to load the table from a persistent data source. See [Section 5.1.7, “Server Command Options”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#server-options), and [Section 13.2.7, “LOAD DATA INFILE Syntax”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#load-data).

**MEMORY Tables and Replication**

A server's MEMORY tables become empty when it is shut down and restarted. If the server is a replication master, its slaves are not aware that these tables have become empty, so you see out-of-date content if you select data from the tables on the slaves. To synchronize master and slave MEMORY tables, when a MEMORY table is used on a master for the first time since it was started, a [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete) statement is written to the master's binary log, to empty the table on the slaves also. The slave still has outdated data in the table during the interval between the master's restart and its first use of the table. To avoid this interval when a direct query to the slave could return stale data, use the [--init-file](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#option_mysqld_init-file) option to populate the MEMORY table on the master at startup.

**Managing Memory Use**

The server needs sufficient memory to maintain all MEMORY tables that are in use at the same time.

Memory is not reclaimed if you delete individual rows from a MEMORY table. Memory is reclaimed only when the entire table is deleted. Memory that was previously used for deleted rows is re-used for new rows within the same table. To free all the memory used by a MEMORY table when you no longer require its contents, execute [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete) or [TRUNCATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#truncate-table) to remove all rows, or remove the table altogether using [DROP TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#drop-table). To free up the memory used by deleted rows, use ALTER TABLE ENGINE=MEMORY to force a table rebuild.

The memory needed for one row in a MEMORY table is calculated using the following expression:

SUM\_OVER\_ALL\_BTREE\_KEYS(*max\_length\_of\_key* + sizeof(char\*) \* 4)

+ SUM\_OVER\_ALL\_HASH\_KEYS(sizeof(char\*) \* 2)

+ ALIGN(*length\_of\_row*+1, sizeof(char\*))

ALIGN() represents a round-up factor to cause the row length to be an exact multiple of the char pointer size. sizeof(char\*) is 4 on 32-bit machines and 8 on 64-bit machines.

As mentioned earlier, the [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) system variable sets the limit on the maximum size of MEMORY tables. To control the maximum size for individual tables, set the session value of this variable before creating each table. (Do not change the global [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) value unless you intend the value to be used for MEMORY tables created by all clients.) The following example creates two MEMORY tables, with a maximum size of 1MB and 2MB, respectively:

mysql> **SET max\_heap\_table\_size = 1024\*1024;**

Query OK, 0 rows affected (0.00 sec)

mysql> **CREATE TABLE t1 (id INT, UNIQUE(id)) ENGINE = MEMORY;**

Query OK, 0 rows affected (0.01 sec)

mysql> **SET max\_heap\_table\_size = 1024\*1024\*2;**

Query OK, 0 rows affected (0.00 sec)

mysql> **CREATE TABLE t2 (id INT, UNIQUE(id)) ENGINE = MEMORY;**

Query OK, 0 rows affected (0.00 sec)

Both tables revert to the server's global [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) value if the server restarts.

You can also specify a MAX\_ROWS table option in [CREATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#create-table) statements for MEMORY tables to provide a hint about the number of rows you plan to store in them. This does not enable the table to grow beyond the [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) value, which still acts as a constraint on maximum table size. For maximum flexibility in being able to use MAX\_ROWS, set [max\_heap\_table\_size](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\server-administration.html#sysvar_max_heap_table_size) at least as high as the value to which you want each MEMORY table to be able to grow.

**Additional Resources**

A forum dedicated to the MEMORY storage engine is available at <https://forums.mysql.com/list.php?92>.

**16.4 The CSV Storage Engine**

[16.4.1 Repairing and Checking CSV Tables](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#se-csv-repair)

[16.4.2 CSV Limitations](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#se-csv-limitations)

The CSV storage engine stores data in text files using comma-separated values format.

The CSV storage engine is always compiled into the MySQL server.

To examine the source for the CSV engine, look in the storage/csv directory of a MySQL source distribution.

When you create a CSV table, the server creates a data file. The data file name begins with the table name and has a .CSV extension. The data file is a plain text file. When you store data into the table, the storage engine saves it into the data file in comma-separated values format.

mysql> **CREATE TABLE test (i INT NOT NULL, c CHAR(10) NOT NULL)**

**ENGINE = CSV;**

Query OK, 0 rows affected (0.06 sec)

mysql> **INSERT INTO test VALUES(1,'record one'),(2,'record two');**

Query OK, 2 rows affected (0.05 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> **SELECT \* FROM test;**

+---+------------+

| i | c |

+---+------------+

| 1 | record one |

| 2 | record two |

+---+------------+

2 rows in set (0.00 sec)

Creating a CSV table also creates a corresponding Metafile that stores the state of the table and the number of rows that exist in the table. The name of this file is the same as the name of the table with the extension CSM.

If you examine the test.CSV file in the database directory created by executing the preceding statements, its contents should look like this:

"1","record one"

"2","record two"

This format can be read, and even written, by spreadsheet applications such as Microsoft Excel or StarOffice Calc.

**16.4.1 Repairing and Checking CSV Tables**

The CSV storage engines supports the CHECK and REPAIR statements to verify and if possible repair a damaged CSV table.

When running the CHECK statement, the CSV file will be checked for validity by looking for the correct field separators, escaped fields (matching or missing quotation marks), the correct number of fields compared to the table definition and the existence of a corresponding CSV metafile. The first invalid row discovered will report an error. Checking a valid table produces output like that shown below:

mysql> **check table csvtest;**

+--------------+-------+----------+----------+

| Table | Op | Msg\_type | Msg\_text |

+--------------+-------+----------+----------+

| test.csvtest | check | status | OK |

+--------------+-------+----------+----------+

1 row in set (0.00 sec)

A check on a corrupted table returns a fault:

mysql> **check table csvtest;**

+--------------+-------+----------+----------+

| Table | Op | Msg\_type | Msg\_text |

+--------------+-------+----------+----------+

| test.csvtest | check | error | Corrupt |

+--------------+-------+----------+----------+

1 row in set (0.01 sec)

If the check fails, the table is marked as crashed (corrupt). Once a table has been marked as corrupt, it is automatically repaired when you next run CHECK or execute a [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) statement. The corresponding corrupt status and new status will be displayed when running CHECK:

mysql> **check table csvtest;**

+--------------+-------+----------+----------------------------+

| Table | Op | Msg\_type | Msg\_text |

+--------------+-------+----------+----------------------------+

| test.csvtest | check | warning | Table is marked as crashed |

| test.csvtest | check | status | OK |

+--------------+-------+----------+----------------------------+

2 rows in set (0.08 sec)

To repair a table you can use REPAIR, this copies as many valid rows from the existing CSV data as possible, and then replaces the existing CSV file with the recovered rows. Any rows beyond the corrupted data are lost.

mysql> **repair table csvtest;**

+--------------+--------+----------+----------+

| Table | Op | Msg\_type | Msg\_text |

+--------------+--------+----------+----------+

| test.csvtest | repair | status | OK |

+--------------+--------+----------+----------+

1 row in set (0.02 sec)

Warning

During repair, only the rows from the CSV file up to the first damaged row are copied to the new table. All other rows from the first damaged row to the end of the table are removed, even valid rows.

**16.4.2 CSV Limitations**

The CSV storage engine does not support indexing.

The CSV storage engine does not support partitioning.

All tables that you create using the CSV storage engine must have the NOT NULL attribute on all columns. However, for backward compatibility, you can continue to use tables with nullable columns that were created in previous MySQL releases. (Bug #32050)

**16.5 The ARCHIVE Storage Engine**

The ARCHIVE storage engine produces special-purpose tables that store large amounts of unindexed data in a very small footprint.

**Table 16.5 ARCHIVE Storage Engine Features**

| **Feature** | **Support** |
| --- | --- |
| **B-tree indexes** | No |
| **Backup/point-in-time recovery** (Implemented in the server, rather than in the storage engine.) | Yes |
| **Cluster database support** | No |
| **Clustered indexes** | No |
| **Compressed data** | Yes |
| **Data caches** | No |
| **Encrypted data** | Yes (Implemented in the server via encryption functions.) |
| **Foreign key support** | No |
| **Full-text search indexes** | No |
| **Geospatial data type support** | Yes |
| **Geospatial indexing support** | No |
| **Hash indexes** | No |
| **Index caches** | No |
| **Locking granularity** | Row |
| **MVCC** | No |
| **Replication support** (Implemented in the server, rather than in the storage engine.) | Yes |
| **Storage limits** | None |
| **T-tree indexes** | No |
| **Transactions** | No |
| **Update statistics for data dictionary** | Yes |

The ARCHIVE storage engine is included in MySQL binary distributions. To enable this storage engine if you build MySQL from source, invoke **CMake** with the [-DWITH\_ARCHIVE\_STORAGE\_ENGINE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\installing.html#option_cmake_storage_engine_options) option.

To examine the source for the ARCHIVE engine, look in the storage/archive directory of a MySQL source distribution.

You can check whether the ARCHIVE storage engine is available with the [SHOW ENGINES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#show-engines) statement.

When you create an ARCHIVE table, the storage engine creates files with names that begin with the table name. The data file has an extension of .ARZ. An .ARN file may appear during optimization operations.

The ARCHIVE engine supports [INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert), [REPLACE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#replace), and [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select), but not [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete) or [UPDATE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#update). It does support ORDER BY operations, [BLOB](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob) columns, and basically all but spatial data types (see [Section 11.5.1, “Spatial Data Types”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#spatial-type-overview)). The ARCHIVE engine uses row-level locking.

The ARCHIVE engine supports the AUTO\_INCREMENT column attribute. The AUTO\_INCREMENT column can have either a unique or nonunique index. Attempting to create an index on any other column results in an error. The ARCHIVE engine also supports the AUTO\_INCREMENT table option in [CREATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#create-table) statements to specify the initial sequence value for a new table or reset the sequence value for an existing table, respectively.

ARCHIVE does not support inserting a value into an AUTO\_INCREMENT column less than the current maximum column value. Attempts to do so result in an [ER\_DUP\_KEY](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\error-handling.html#error_er_dup_key) error.

The ARCHIVE engine ignores [BLOB](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\data-types.html#blob) columns if they are not requested and scans past them while reading.

The ARCHIVE storage engine does not support partitioning.

**Storage:** Rows are compressed as they are inserted. The ARCHIVE engine uses zlib lossless data compression (see <http://www.zlib.net/>). You can use [OPTIMIZE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#optimize-table) to analyze the table and pack it into a smaller format (for a reason to use [OPTIMIZE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#optimize-table), see later in this section). The engine also supports [CHECK TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#check-table). There are several types of insertions that are used:

* An [INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert) statement just pushes rows into a compression buffer, and that buffer flushes as necessary. The insertion into the buffer is protected by a lock. A [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) forces a flush to occur.
* A bulk insert is visible only after it completes, unless other inserts occur at the same time, in which case it can be seen partially. A [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) never causes a flush of a bulk insert unless a normal insert occurs while it is loading.

**Retrieval**: On retrieval, rows are uncompressed on demand; there is no row cache. A [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) operation performs a complete table scan: When a [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) occurs, it finds out how many rows are currently available and reads that number of rows. [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) is performed as a consistent read. Note that lots of [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) statements during insertion can deteriorate the compression, unless only bulk inserts are used. To achieve better compression, you can use [OPTIMIZE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#optimize-table) or [REPAIR TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#repair-table). The number of rows in ARCHIVE tables reported by [SHOW TABLE STATUS](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#show-table-status) is always accurate. See [Section 13.7.3.4, “OPTIMIZE TABLE Syntax”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#optimize-table), [Section 13.7.3.5, “REPAIR TABLE Syntax”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#repair-table), and [Section 13.7.6.36, “SHOW TABLE STATUS Syntax”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#show-table-status).

**Additional Resources**

* A forum dedicated to the ARCHIVE storage engine is available at <https://forums.mysql.com/list.php?112>.

**16.6 The BLACKHOLE Storage Engine**

The BLACKHOLE storage engine acts as a “black hole” that accepts data but throws it away and does not store it. Retrievals always return an empty result:

mysql> **CREATE TABLE test(i INT, c CHAR(10)) ENGINE = BLACKHOLE;**

Query OK, 0 rows affected (0.03 sec)

mysql> **INSERT INTO test VALUES(1,'record one'),(2,'record two');**

Query OK, 2 rows affected (0.00 sec)

Records: 2 Duplicates: 0 Warnings: 0

mysql> **SELECT \* FROM test;**

Empty set (0.00 sec)

To enable the BLACKHOLE storage engine if you build MySQL from source, invoke **CMake** with the [-DWITH\_BLACKHOLE\_STORAGE\_ENGINE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\installing.html#option_cmake_storage_engine_options) option.

To examine the source for the BLACKHOLE engine, look in the sql directory of a MySQL source distribution.

When you create a BLACKHOLE table, the server creates the table definition in the global data dictionary. There are no files associated with the table.

The BLACKHOLE storage engine supports all kinds of indexes. That is, you can include index declarations in the table definition.

The BLACKHOLE storage engine does not support partitioning.

You can check whether the BLACKHOLE storage engine is available with the [SHOW ENGINES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#show-engines) statement.

Inserts into a BLACKHOLE table do not store any data, but if statement based binary logging is enabled, the SQL statements are logged and replicated to slave servers. This can be useful as a repeater or filter mechanism.

Suppose that your application requires slave-side filtering rules, but transferring all binary log data to the slave first results in too much traffic. In such a case, it is possible to set up on the master host a “dummy” slave process whose default storage engine is BLACKHOLE, depicted as follows:

**Figure 16.1 Replication using BLACKHOLE for Filtering**

The master writes to its binary log. The “dummy” [**mysqld**](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\programs.html#mysqld) process acts as a slave, applying the desired combination of replicate-do-\* and replicate-ignore-\* rules, and writes a new, filtered binary log of its own. (See [Section 17.1.6, “Replication and Binary Logging Options and Variables”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\replication.html#replication-options).) This filtered log is provided to the slave.

The dummy process does not actually store any data, so there is little processing overhead incurred by running the additional [**mysqld**](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\programs.html#mysqld) process on the replication master host. This type of setup can be repeated with additional replication slaves.

[INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert) triggers for BLACKHOLE tables work as expected. However, because the BLACKHOLE table does not actually store any data, [UPDATE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#update) and [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete) triggers are not activated: The FOR EACH ROW clause in the trigger definition does not apply because there are no rows.

Other possible uses for the BLACKHOLE storage engine include:

* Verification of dump file syntax.
* Measurement of the overhead from binary logging, by comparing performance using BLACKHOLE with and without binary logging enabled.
* BLACKHOLE is essentially a “no-op” storage engine, so it could be used for finding performance bottlenecks not related to the storage engine itself.

The BLACKHOLE engine is transaction-aware, in the sense that committed transactions are written to the binary log and rolled-back transactions are not.

**Blackhole Engine and Auto Increment Columns**

The Blackhole engine is a no-op engine. Any operations performed on a table using Blackhole will have no effect. This should be born in mind when considering the behavior of primary key columns that auto increment. The engine will not automatically increment field values, and does not retain auto increment field state. This has important implications in replication.

Consider the following replication scenario where all three of the following conditions apply:

1. On a master server there is a blackhole table with an auto increment field that is a primary key.
2. On a slave the same table exists but using the MyISAM engine.
3. Inserts are performed into the master's table without explicitly setting the auto increment value in the INSERT statement itself or through using a SET INSERT\_ID statement.

In this scenario replication will fail with a duplicate entry error on the primary key column.

In statement based replication, the value of INSERT\_ID in the context event will always be the same. Replication will therefore fail due to trying insert a row with a duplicate value for a primary key column.

In row based replication, the value that the engine returns for the row always be the same for each insert. This will result in the slave attempting to replay two insert log entries using the same value for the primary key column, and so replication will fail.

**Column Filtering**

When using row-based replication, ([binlog\_format=ROW](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\replication.html#sysvar_binlog_format)), a slave where the last columns are missing from a table is supported, as described in the section [Section 17.4.1.9, “Replication with Differing Table Definitions on Master and Slave”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\replication.html#replication-features-differing-tables).

This filtering works on the slave side, that is, the columns are copied to the slave before they are filtered out. There are at least two cases where it is not desirable to copy the columns to the slave:

1. If the data is confidential, so the slave server should not have access to it.
2. If the master has many slaves, filtering before sending to the slaves may reduce network traffic.

Master column filtering can be achieved using the BLACKHOLE engine. This is carried out in a way similar to how master table filtering is achieved - by using the BLACKHOLE engine and the [--replicate-do-table](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\replication.html#option_mysqld_replicate-do-table) or [--replicate-ignore-table](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\replication.html#option_mysqld_replicate-ignore-table) option.

The setup for the master is:

CREATE TABLE t1 (public\_col\_1, ..., public\_col\_N,

secret\_col\_1, ..., secret\_col\_M) ENGINE=MyISAM;

The setup for the trusted slave is:

CREATE TABLE t1 (public\_col\_1, ..., public\_col\_N) ENGINE=BLACKHOLE;

The setup for the untrusted slave is:

CREATE TABLE t1 (public\_col\_1, ..., public\_col\_N) ENGINE=MyISAM;

**16.7 The MERGE Storage Engine**

[16.7.1 MERGE Table Advantages and Disadvantages](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#merge-table-advantages)

[16.7.2 MERGE Table Problems](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\storage-engines.html#merge-table-problems)

The MERGE storage engine, also known as the MRG\_MyISAM engine, is a collection of identical MyISAM tables that can be used as one. “Identical” means that all tables have identical column data types and index information. You cannot merge MyISAM tables in which the columns are listed in a different order, do not have exactly the same data types in corresponding columns, or have the indexes in different order. However, any or all of the MyISAM tables can be compressed with [**myisampack**](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\programs.html#myisampack). See [Section 4.6.6, “**myisampack** — Generate Compressed, Read-Only MyISAM Tables”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\programs.html#myisampack). Differences between tables such as these do not matter:

* Names of corresponding columns and indexes can differ.
* Comments for tables, columns, and indexes can differ.
* Table options such as AVG\_ROW\_LENGTH, MAX\_ROWS, or PACK\_KEYS can differ.

An alternative to a MERGE table is a partitioned table, which stores partitions of a single table in separate files and enables some operations to be performed more efficiently. For more information, see [Chapter 23, *Partitioning*](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\partitioning.html).

When you create a MERGE table, MySQL creates a .MRG file on disk that contains the names of the underlying MyISAM tables that should be used as one. The table format of the MERGE table is stored in the MySQL data dictionary. The underlying tables do not have to be in the same database as the MERGE table.

You can use [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select), [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete), [UPDATE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#update), and [INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert) on MERGE tables. You must have [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\security.html#priv_select), [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\security.html#priv_delete), and [UPDATE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\security.html#priv_update) privileges on the MyISAM tables that you map to a MERGE table.

Note

The use of MERGE tables entails the following security issue: If a user has access to MyISAM table *t*, that user can create a MERGE table *m* that accesses *t*. However, if the user's privileges on *t* are subsequently revoked, the user can continue to access *t* by doing so through *m*.

Use of [DROP TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#drop-table) with a MERGE table drops only the MERGE specification. The underlying tables are not affected.

To create a MERGE table, you must specify a UNION=(*list-of-tables*) option that indicates which MyISAM tables to use. You can optionally specify an INSERT\_METHOD option to control how inserts into the MERGE table take place. Use a value of FIRST or LAST to cause inserts to be made in the first or last underlying table, respectively. If you specify no INSERT\_METHOD option or if you specify it with a value of NO, inserts into the MERGE table are not permitted and attempts to do so result in an error.

The following example shows how to create a MERGE table:

mysql> **CREATE TABLE t1 (**

-> **a INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,**

-> **message CHAR(20)) ENGINE=MyISAM;**

mysql> **CREATE TABLE t2 (**

-> **a INT NOT NULL AUTO\_INCREMENT PRIMARY KEY,**

-> **message CHAR(20)) ENGINE=MyISAM;**

mysql> **INSERT INTO t1 (message) VALUES ('Testing'),('table'),('t1');**

mysql> **INSERT INTO t2 (message) VALUES ('Testing'),('table'),('t2');**

mysql> **CREATE TABLE total (**

-> **a INT NOT NULL AUTO\_INCREMENT,**

-> **message CHAR(20), INDEX(a))**

-> **ENGINE=MERGE UNION=(t1,t2) INSERT\_METHOD=LAST;**

Column a is indexed as a PRIMARY KEY in the underlying MyISAM tables, but not in the MERGE table. There it is indexed but not as a PRIMARY KEY because a MERGE table cannot enforce uniqueness over the set of underlying tables. (Similarly, a column with a UNIQUE index in the underlying tables should be indexed in the MERGE table but not as a UNIQUE index.)

After creating the MERGE table, you can use it to issue queries that operate on the group of tables as a whole:

mysql> **SELECT \* FROM total;**

+---+---------+

| a | message |

+---+---------+

| 1 | Testing |

| 2 | table |

| 3 | t1 |

| 1 | Testing |

| 2 | table |

| 3 | t2 |

+---+---------+

To remap a MERGE table to a different collection of MyISAM tables, you can use one of the following methods:

* DROP the MERGE table and re-create it.
* Use ALTER TABLE *tbl\_name* UNION=(...) to change the list of underlying tables.

It is also possible to use ALTER TABLE ... UNION=() (that is, with an empty [UNION](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#union) clause) to remove all of the underlying tables. However, in this case, the table is effectively empty and inserts fail because there is no underlying table to take new rows. Such a table might be useful as a template for creating new MERGE tables with [CREATE TABLE ... LIKE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#create-table-like).

The underlying table definitions and indexes must conform closely to the definition of the MERGE table. Conformance is checked when a table that is part of a MERGE table is opened, not when the MERGE table is created. If any table fails the conformance checks, the operation that triggered the opening of the table fails. This means that changes to the definitions of tables within a MERGE may cause a failure when the MERGE table is accessed. The conformance checks applied to each table are:

* The underlying table and the MERGE table must have the same number of columns.
* The column order in the underlying table and the MERGE table must match.
* Additionally, the specification for each corresponding column in the parent MERGE table and the underlying tables are compared and must satisfy these checks:
  + The column type in the underlying table and the MERGE table must be equal.
  + The column length in the underlying table and the MERGE table must be equal.
  + The column of the underlying table and the MERGE table can be NULL.
* The underlying table must have at least as many indexes as the MERGE table. The underlying table may have more indexes than the MERGE table, but cannot have fewer.

Note

A known issue exists where indexes on the same columns must be in identical order, in both the MERGE table and the underlying MyISAM table. See Bug #33653.

Each index must satisfy these checks:

* + The index type of the underlying table and the MERGE table must be the same.
  + The number of index parts (that is, multiple columns within a compound index) in the index definition for the underlying table and the MERGE table must be the same.
  + For each index part:
    - Index part lengths must be equal.
    - Index part types must be equal.
    - Index part languages must be equal.
    - Check whether index parts can be NULL.

If a MERGE table cannot be opened or used because of a problem with an underlying table, [CHECK TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#check-table) displays information about which table caused the problem.

**Additional Resources**

* A forum dedicated to the MERGE storage engine is available at <https://forums.mysql.com/list.php?93>.

**16.7.1 MERGE Table Advantages and Disadvantages**

MERGE tables can help you solve the following problems:

* Easily manage a set of log tables. For example, you can put data from different months into separate tables, compress some of them with [**myisampack**](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\programs.html#myisampack), and then create a MERGE table to use them as one.
* Obtain more speed. You can split a large read-only table based on some criteria, and then put individual tables on different disks. A MERGE table structured this way could be much faster than using a single large table.
* Perform more efficient searches. If you know exactly what you are looking for, you can search in just one of the underlying tables for some queries and use a MERGE table for others. You can even have many different MERGE tables that use overlapping sets of tables.
* Perform more efficient repairs. It is easier to repair individual smaller tables that are mapped to a MERGE table than to repair a single large table.
* Instantly map many tables as one. A MERGE table need not maintain an index of its own because it uses the indexes of the individual tables. As a result, MERGE table collections are *very* fast to create or remap. (You must still specify the index definitions when you create a MERGE table, even though no indexes are created.)
* If you have a set of tables from which you create a large table on demand, you can instead create a MERGE table from them on demand. This is much faster and saves a lot of disk space.
* Exceed the file size limit for the operating system. Each MyISAM table is bound by this limit, but a collection of MyISAM tables is not.
* You can create an alias or synonym for a MyISAM table by defining a MERGE table that maps to that single table. There should be no really notable performance impact from doing this (only a couple of indirect calls and memcpy() calls for each read).

The disadvantages of MERGE tables are:

* You can use only identical MyISAM tables for a MERGE table.
* Some MyISAM features are unavailable in MERGE tables. For example, you cannot create FULLTEXT indexes on MERGE tables. (You can create FULLTEXT indexes on the underlying MyISAM tables, but you cannot search the MERGE table with a full-text search.)
* If the MERGE table is nontemporary, all underlying MyISAM tables must be nontemporary. If the MERGE table is temporary, the MyISAM tables can be any mix of temporary and nontemporary.
* MERGE tables use more file descriptors than MyISAM tables. If 10 clients are using a MERGE table that maps to 10 tables, the server uses (10 × 10) + 10 file descriptors. (10 data file descriptors for each of the 10 clients, and 10 index file descriptors shared among the clients.)
* Index reads are slower. When you read an index, the MERGE storage engine needs to issue a read on all underlying tables to check which one most closely matches a given index value. To read the next index value, the MERGE storage engine needs to search the read buffers to find the next value. Only when one index buffer is used up does the storage engine need to read the next index block. This makes MERGE indexes much slower on [eq\_ref](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#jointype_eq_ref) searches, but not much slower on [ref](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#jointype_ref) searches. For more information about [eq\_ref](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#jointype_eq_ref) and [ref](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\optimization.html#jointype_ref), see [Section 13.8.2, “EXPLAIN Syntax”](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#explain).

**16.7.2 MERGE Table Problems**

The following are known problems with MERGE tables:

* In versions of MySQL Server prior to 5.1.23, it was possible to create temporary merge tables with nontemporary child MyISAM tables.

From versions 5.1.23, MERGE children were locked through the parent table. If the parent was temporary, it was not locked and so the children were not locked either. Parallel use of the MyISAM tables corrupted them.

* If you use [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table) to change a MERGE table to another storage engine, the mapping to the underlying tables is lost. Instead, the rows from the underlying MyISAM tables are copied into the altered table, which then uses the specified storage engine.
* The INSERT\_METHOD table option for a MERGE table indicates which underlying MyISAM table to use for inserts into the MERGE table. However, use of the AUTO\_INCREMENT table option for that MyISAM table has no effect for inserts into the MERGE table until at least one row has been inserted directly into the MyISAM table.
* A MERGE table cannot maintain uniqueness constraints over the entire table. When you perform an [INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert), the data goes into the first or last MyISAM table (as determined by the INSERT\_METHOD option). MySQL ensures that unique key values remain unique within that MyISAM table, but not over all the underlying tables in the collection.
* Because the MERGE engine cannot enforce uniqueness over the set of underlying tables, [REPLACE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#replace) does not work as expected. The two key facts are:
  + [REPLACE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#replace) can detect unique key violations only in the underlying table to which it is going to write (which is determined by the INSERT\_METHOD option). This differs from violations in the MERGE table itself.
  + If [REPLACE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#replace) detects a unique key violation, it will change only the corresponding row in the underlying table it is writing to; that is, the first or last table, as determined by the INSERT\_METHOD option.

Similar considerations apply for [INSERT ... ON DUPLICATE KEY UPDATE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert-on-duplicate).

* MERGE tables do not support partitioning. That is, you cannot partition a MERGE table, nor can any of a MERGE table's underlying MyISAM tables be partitioned.
* You should not use [ANALYZE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#analyze-table), [REPAIR TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#repair-table), [OPTIMIZE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#optimize-table), [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table), [DROP TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#drop-table), [DELETE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#delete) without a WHERE clause, or [TRUNCATE TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#truncate-table) on any of the tables that are mapped into an open MERGE table. If you do so, the MERGE table may still refer to the original table and yield unexpected results. To work around this problem, ensure that no MERGE tables remain open by issuing a [FLUSH TABLES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#flush-tables) statement prior to performing any of the named operations.

The unexpected results include the possibility that the operation on the MERGE table will report table corruption. If this occurs after one of the named operations on the underlying MyISAM tables, the corruption message is spurious. To deal with this, issue a [FLUSH TABLES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#flush-tables) statement after modifying the MyISAM tables.

* [DROP TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#drop-table) on a table that is in use by a MERGE table does not work on Windows because the MERGE storage engine's table mapping is hidden from the upper layer of MySQL. Windows does not permit open files to be deleted, so you first must flush all MERGE tables (with [FLUSH TABLES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#flush-tables)) or drop the MERGE table before dropping the table.
* The definition of the MyISAM tables and the MERGE table are checked when the tables are accessed (for example, as part of a [SELECT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#select) or [INSERT](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#insert) statement). The checks ensure that the definitions of the tables and the parent MERGE table definition match by comparing column order, types, sizes and associated indexes. If there is a difference between the tables, an error is returned and the statement fails. Because these checks take place when the tables are opened, any changes to the definition of a single table, including column changes, column ordering, and engine alterations will cause the statement to fail.
* The order of indexes in the MERGE table and its underlying tables should be the same. If you use [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table) to add a UNIQUE index to a table used in a MERGE table, and then use [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table) to add a nonunique index on the MERGE table, the index ordering is different for the tables if there was already a nonunique index in the underlying table. (This happens because [ALTER TABLE](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#alter-table) puts UNIQUE indexes before nonunique indexes to facilitate rapid detection of duplicate keys.) Consequently, queries on tables with such indexes may return unexpected results.
* If you encounter an error message similar to ERROR 1017 (HY000): Can't find file: '*tbl\_name*.MRG' (errno: 2), it generally indicates that some of the underlying tables do not use the MyISAM storage engine. Confirm that all of these tables are MyISAM.
* The maximum number of rows in a MERGE table is 264 (~1.844E+19; the same as for a MyISAM table). It is not possible to merge multiple MyISAM tables into a single MERGE table that would have more than this number of rows.
* Use of underlying MyISAM tables of differing row formats with a parent MERGE table is currently known to fail. See Bug #32364.
* You cannot change the union list of a nontemporary MERGE table when [LOCK TABLES](file:///C:\Users\qxz\AppData\Local\Temp\Rar$EXa0.665\sql-syntax.html#lock-tables) is in effect. The following does *not* work:
* CREATE TABLE m1 ... ENGINE=MRG\_MYISAM ...;
* LOCK TABLES t1 WRITE, t2 WRITE, m1 WRITE;
* ALTER TABLE m1 ... UNION=(t1,t2) ...;

However, you can do this with a temporary MERGE table.

* You cannot create a MERGE table with CREATE ... SELECT, neither as a temporary MERGE table, nor as a nontemporary MERGE table. For example:

CREATE TABLE m1 ... ENGINE=MRG\_MYISAM ... SELECT ...;

Attempts to do this result in an error: *tbl\_name* is not BASE TABLE.

* In some cases, differing PACK\_KEYS table option values among the MERGE and underlying tables cause unexpected results if the underlying tables contain CHAR or BINARY columns. As a workaround, use ALTER TABLE to ensure that all involved tables have the same PACK\_KEYS value. (Bug #50646)