16.3 内存存储引擎

内存存储引擎（以前称为堆）使用存储在内存中的内容创建特殊用途的表。由于数据易受崩溃、硬件问题或断电的影响，因此只能将这些表用作临时工作区或从其他表中提取的数据的只读缓存。

16.4 内存存储引擎特性

表 16-4-1

特性	支持
树索引	是
备份/时间点恢复（在服务器中实现，而不是在存储引擎中实现。）	是
群集数据库支持	否
群集索引	否
压缩数据	否
数据缓存	N/A
数据加密传输	是
外键支持	否
全文搜索索引	否
地理空间数据类型支持	否
地理空间索引支持	否
哈希索引	是
索引缓存	不适用
锁定粒度	表
MVCC	否
复制支持（在服务器中实现，而不是在存储引擎中实现）有限	（请参阅本节后面的讨论）
存储限制	RAM
T 树索引	否
交易	否
更新数据字典的统计信息	是

何时使用内存或 NDB 集群

如果开发人员希望部署使用内存存储引擎来存储重要、高可用或频繁更新的数据的应用程序，则应考虑 NDB 集群是否是一个更好的选择。内存引擎的典型用例包含以下特性：

•涉及瞬态、非关键数据的操作，如会话管理或缓存。当 MySQL 服务器停止或重新启动时，内存表中的数据将丢失。

•内存存储，实现快速访问和低延迟。数据卷可以完全放在内存中，而不会导致操作系统交换虚拟内存页。

•只读或以读为主的数据访问模式（有限更新）。

何时使用内存或 NDB 集群

如果开发人员希望部署使用内存存储引擎来存储重要、高可用或频繁更新的数据的应用程序，则应考虑 NDB 集群是否是一个更好的选择。内存引擎的典型用例包含以下特性：

•涉及瞬态、非关键数据的操作，如会话管理或缓存。当 MySQL 服务器停止或重新启动时，内存表中的数据将丢失。

•内存存储，实现快速访问和低延迟。数据卷可以完全放在内存中，而不会导致操作系统交换虚拟内存页。

•只读或以读为主的数据访问模式（有限更新）。

分区

内存表不能分区。

性能特点

内存性能受单线程执行导致的争用和处理更新时的表锁开销限制。这限制了负载增加时的可伸缩性，特别是对于包含写操作的语句混合。

尽管对内存表进行了内存处理，但对于通用查询或读/写工作负载，它们并不一定比繁忙服务器上的 InnoDB 表快。特别是，执行更新所涉及的表锁定可以减缓多个会话中内存表的并发使用。

根据对内存表执行的查询类型，可以将索引创建为默认哈希数据结构（用于基于唯一键查找单个值）或通用 B 树数据结构（用于涉及等式、不等式或范围运算符（如小于或大于）的所有查询）。以下各节说明创建这两种索引的语法。一个常见的性能问题是在 B 树索引效率更高的工作负载中使用默认哈希索引。

内存表的特性

内存存储引擎不会在磁盘上创建任何文件。表定义存储在 MySQL 数据字典中。

内存表具有以下特征：

•内存表的空间是以小块的形式分配的。表对插入使用 100%动态哈希。不需要溢出区域或额外的密钥空间。免费列表不需要额外的空间。删除的行放在链接列表中，并在向表中插入新数据时重复使用。内存表也没有通常与哈希表中的 delete 和 insert 相关联的问题。

•内存表使用固定长度的行存储格式。可变长度类型（如 VARCHAR）使用固定长度存储。

•内存表不能包含 BLOB 或文本列。

•内存包括对自动递增列的支持。

•所有客户机之间共享非临时内存表，就像任何其他非临时表一样。

内存表的 DDL 操作

要创建内存表，请在 CREATETABLE 语句中指定 ENGINE=MEMORY 子句。

创建表 t（i INT）ENGINE=MEMORY；

如引擎名所示，内存表存储在内存中。默认情况下，它们使用散列索引，这使得它们对于单值查找非常快速，并且对于创建临时表非常有用。但是，当服务器关闭时，存储在内存表中的所有行都将丢失。

表本身仍然存在，因为它们的定义存储在 MySQL 数据字典中，但在服务器重新启动时它们是空的。

此示例显示如何创建、使用和删除内存表：

mysql>创建表测试引擎=内存

选择 ip，SUM（下载）为 down

按 ip 从日志表组；

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

mysql>删除表测试；

内存表的最大大小受 Max　HeaPaTable 大小系统变量的限制，该变量具有 16MB 的默认值。要对内存表强制不同的大小限制，请更改此变量的值。对于 CREATE　TABLE、后续 ALTER TABLE 或 TRUNCATE TABLE 有效的值是用于表生命周期的值。服务器重新启动还将现有内存表的最大大小设置为全局 Max　HypAppTable 大小的值。您可以设置单个表的大小，如本节后面所述。

索引

内存存储引擎支持哈希和 BTREE 索引。通过添加 USING 子句，可以为给定索引指定一个或另一个，如下所示：

```
创建表查找
(id INT，使用哈希(id)的索引)
引擎=存储器；
创建表查找(id INT，使用 BTREE(id)的索引)
引擎=存储器；
```

有关 B 树和哈希索引的一般特性，请参阅第 8.3.1 节“MySQL 如何使用索引”。

内存表每表最多可有 64 个索引，每个索引有 16 个列，最大的密钥长度为 3072 个字节。

如果内存表哈希索引具有高度的键重复(许多索引项包含相同的值)，则对影响键值和所有删除的表的更新速度会明显减慢。这种放缓的程度与重复程度成正比(或与指数基数成反比)。可以使用 BTREE 索引来避免此问题。

内存表可以有非唯一键。(对于哈希索引的实现，这是一个不常见的特性。)

索引的列可以包含空值。

用户创建的表和临时表

内存表内容存储在内存中，这是内存表与服务器在处理查询时动态创建的内部临时表共享的属性。但是，这两种表的不同之处在于，内存表不受存储转换的影响，而内部临时表是：

- 如果内部临时表太大，服务器会自动将其转换为磁盘存储，如第 8.4.4 节“MySQL 中的内部临时表使用”所述。
- 用户创建的内存表永远不会转换为磁盘表。

加载数据

要在 MySQL 服务器启动时填充内存表，可以使用--init file 选项。例如，可以将 INSERT 等语句放入…选择或将数据内嵌到此文件中以从永久数据源加载表。请参阅第 5.1.7 节“服务器命令选项”和第 13.2.7 节“加载数据填充语法”。

内存表和复制

服务器的内存表在关闭和重新启动时变为空。如果服务器是复制主服务器，则其从属服务器不会意识到这些表已变为空，因此如果从从属服务器上的表中选择数据，则会看到过期的内容。要同步主内存表和从内存表，当主内存表自启动以来第一次在主内存表上使用时，会将 DELETE 语句写入主内存的二进制日志，同时清空从内存表。从机在主机重新启动和首次使用表之间的时间间隔内，表中仍有过时的数据。若要避免直接查询从机可能返回过时数据时出现此间隔，请在启动时使用--init file 选项填充主机上的内存表。

管理内存使用

服务器需要足够的内存来维护同时使用的所有内存表。
如果从内存表中删除单个行，则不会回收内存。只有删除整个表时，才会回收内存。以前用于删除行的内存将重新用于同一表中的新行。要在不再需要内存表的内容时释放内存表使用的所有内存，请执行 DELETE 或 TRUNCATE table 来删除所有行，
或者使用 DROP　table 完全删除表。要释放已删除行使用的内存，请使用 ALTER　TABLE ENGINE=memory 强制表重建。
使用以下表达式计算内存表中一行所需的内存：
SUM_OVER_ALL_BTREE_key（密钥的最大长度+sizeof（char*）*4）
+所有哈希键之和（sizeof（char*）*2）
+对齐（行的长度+1，大小（字符*）
ALIGN（）表示使行长度为字符指针大小的精确倍数的舍入因子。sizeof（char*）在 32 位计算机上为 4，在 64 位计算机上为 8。
正如前面提到的，Max　HeaPaTabLayLyStand 系统变量设置了内存表最大大小的限制。若要控制各个表的最大大小，请在创建每个表之前设置此变量的会话值。除非您想将值用于所有客户端创建的内存表中，否则不更改全局 Max　HeavaPabl 大小）。下面的示例分别创建两个内存表，最大大小分别为 1MB 和 2MB：
mysql>SET max_heap_table_size=1024*1024；
查询正常，0 行受影响（0.00 秒）
mysql>CREATE TABLE t1（id INT，UNIQUE（id））ENGINE=MEMORY；
查询正常，0 行受影响（0.01 秒）
mysql>设置最大堆表大小=1024*1024*2；
查询正常，0 行受影响（0.00 秒）
mysql>CREATE TABLE t2（id INT，UNIQUE（id））ENGINE=MEMORY；
查询正常，0 行受影响（0.00 秒）
如果服务器重新启动，两个表都将还原为服务器的全局最大堆表大小值。
还可以在内存表的 CREATE table 语句中指定 MAX U ROWS table 选项，以提供有关计划存储在其中的行数的提示。这并不能使表超出 Max　HeAPPabTable 大小的值，这仍然是对最大表大小的限制。为了能够最大限度地使用 Max 行，请将 Max　THEAPAPTable 大小设置为与每个内存表能够增长的值一样高。

额外资源

在 https://forums.mysql.com/list.php 上有一个专门讨论内存存储引擎的论坛？92 年。

16.4 CSV 存储引擎

16.4.1 修复和检查 CSV 表格
16.4.2 CSV 限制
CSV 存储引擎使用逗号分隔值格式将数据存储在文本文件中。
CSV 存储引擎总是编译到 MySQL 服务器中。

要检查 CSV 引擎的源代码，请查看 MySQL 源代码发行版的 storage/CSV 目录。
创建 CSV 表时，服务器将创建数据文件。数据文件名以表名开头，扩展名为.CSV。数据文件是纯文本文件。将数据存储到表中时，存储引擎将其以逗号分隔值格式保存到数据文件中。
mysql>创建表测试(i INT 不为空，c CHAR(10)不为空)
引擎=CSV;
查询正常，0 行受影响(0.06 秒)
mysql>插入测试值(1，'record one'),(2，'record two');
查询正常，2 行受影响(0.05 秒)
记录:2 重复:0 警告:0
mysql>从测试中选择*;
```
+---+------------+
|I       | c 公司|
+---+------------+
|1     |记录 1|
|2     |记录 2|
+---+------------+
```
一组 2 行(0.00 秒)
创建 CSV 表还创建一个对应的元文件，它存储表的状态和表中存在的行数。此文件的名称与扩展名为 CSM 的表的名称相同。
如果检查通过执行上述语句创建的数据库目录中的 test.CSV 文件，其内容应如下所示:
“1”,“记录 1”
“2”,“记录 2”
这种格式可以由电子表格应用程序(如 Microsoft Excel 或 StarOffice Calc)读取甚至写入。

16.4.1 修复和检查 CSV 表格

CSV 存储引擎支持 CHECK 和 REPAIR 语句来验证并在可能的情况下修复损坏的 CSV 表。运行 CHECK 语句时，将通过查找正确的字段分隔符、转义字段(匹配或丢失引号)、与表定义相比的正确字段数以及相应的 CSV 元文件的存在来检查 CSV 文件的有效性。发现的第一个无效行将报告错误。检查有效表会产生如下所示的输出:
mysql>检查表 csvtest;
```
+--------------+-------+----------+----------+
|表|操作|消息|类型|消息|文本|
+--------------+-------+----------+----------+
|test.csvtest |检查|状态|正常|
+--------------+-------+----------+----------+
```
一行(0.00 秒)
对损坏表的检查返回错误:
mysql>检查表 csvtest;
```
+--------------+-------+----------+----------+
|表|操作|消息|类型|消息|文本|
+--------------+-------+----------+----------+
|test.csvtest |检查|错误|损坏|
+--------------+-------+----------+----------+
```

一行一组(0.01 秒)
如果检查失败，则将表标记为崩溃(损坏)。一旦表被标记为已损坏，则在下次运行 CHECK 或执行 SELECT 语句时会自动修复该表。运行检查时，将显示相应的损坏状态和新状态:
mysql>检查表 csvtest;
+--------------+-------+----------+--------------------------+
|表|操作|消息|类型|消息|文本|
+--------------+-------+----------+--------------------------+
|test.csvtest | check | warning |表被标记为崩溃|
|test.csvtest |检查|状态|正常|
+--------------+-------+----------+--------------------------+
一组 2 行(0.08 秒)
若要修复表，可以使用修复，它将尽可能多地从现有 CSV 数据复制有效行，然后用恢复的行替换现有 CSV 文件。损坏数据之外的任何行都将丢失。
mysql>修复表 csvtest;
+--------------+--------+----------+----------+
|表|操作|消息|类型|消息|文本|
+--------------+--------+----------+----------+
|test.csvtest |修复|状态|正常|
+--------------+--------+----------+----------+
一行一组(0.02 秒)
警告
在修复过程中，只有从 CSV 文件到第一个损坏行的行被复制到新表中。从第一个损坏行到表末尾的所有其他行都将被删除，即使是有效行。

16.4.2 CSV 限制

CSV 存储引擎不支持索引。
CSV 存储引擎不支持分区。
使用 CSV 存储引擎创建的所有表在所有列上都必须具有 NOT NULL 属性。但是，为了向后兼容，
您可以继续使用在以前的 MySQL 版本中创建的具有可空列的表。(错误#32050)

16.5 档案存储引擎

存档存储引擎生成特殊用途的表，这些表将大量未编制索引的数据存储在非常小的内存中。
表 16.5 存档存储引擎

特性	功能支持
B-树索引号	否
备份/时间点恢复(在服务器中实现，而不是在存储引擎中实现)	是
群集数据库支持	否
聚集索引	否
压缩数据	是
数据缓存	否

加密数据(通过加密功能在服务器中实现)	是
外键支持	否
全文搜索索引	否
地理空间数据类型支持	是
地理空间索引	支持
哈希索引	否
索引缓存编号	否
锁定粒度行 MVCC	否 否
复制支持(在服务器中实现，而不是在存储引擎中实现)	是
无存储限制	无
T-树索引号	否
交易编号	否
更新数据字典的统计信息	是

存档存储引擎包含在 MySQL 二进制发行版中。若要在从源代码构建 MySQL 时启用此存储引擎，请使用-DWITH_ARCHIVE_storage_engine 选项调用 CMake。
要检查存档引擎的源代码，请查看 MySQL 源发行版的 storage/ARCHIVE 目录。
您可以使用 SHOW ENGINES 语句检查存档存储引擎是否可用。
创建存档表时，存储引擎将创建名称以表名开头的文件。数据文件的扩展名为.ARZ。优化操作期间可能会出现.ARN 文件。
存档引擎支持插入、替换和选择，但不支持删除或更新。它确实支持按顺序操作、BLOB 列和基本上除空间数据类型之外的所有类型(请参阅第 11.5.1 节“空间数据类型”)。存档引擎使用行级锁定。
存档引擎支持自动递增列属性。“自动增量”列可以具有唯一索引或非唯一索引。试图在任何其他列上创建索引将导致错误。归档引擎还支持 CREATETABLE 语句中的 AutoYLoad 表选项，以指定新表的初始序列值或重置现有表的序列值。
存档不支持将值插入到小于当前最大列值的 AutoYLoad 列中。尝试执行此操作会导致重复密钥错误。
如果没有请求 BLOB 列，存档引擎将忽略它们，并在读取时扫描它们。

存档存储引擎不支持分区。

存储:行在插入时被压缩。存档引擎使用 zlib 无损数据压缩(请参见 http://www.zlib.net/)。您可以使用 OPTIMIZE TABLE 来分析表并将其打包成较小的格式(有关使用 OPTIMIZE TABLE 的原因，请参阅本节后面的内容)。发动机还支持检查台。有几种类型的插入使用:
•INSERT 语句只是将行推入压缩缓冲区，缓冲区会根据需要进行刷新。插入缓冲区受锁保护。选择将强制发生刷新。
•大容量插入只有在完成后才可见，除非其他插入同时发生，在这种情况下，可以部分看到。除非加载时发生正常插入，否则 SELECT 永远不会导致大容量插入的刷新。
检索:检索时，行按需解压缩;没有行缓存。SELECT 操作执行一个完整的表扫描:当 SELECT 发生时，它会发现当前有多少行可用，并读取该行数。选择作为一致读取执行。

请注意，除非只使用大容量插入，否则插入过程中的许多 SELECT 语句可能会恶化压缩。为了获得更好的压缩，可以使用优化表或修复表。“显示表状态”报告的存档表中的行数始终准确。请参阅第 13.7.3.4 节“优化表语法”、第 13.7.3.5 节“修复表语法”和第 13.7.6.36 节“显示表状态语法”。

额外资源

•在 https://forums.mysql.com/list.php 上有一个专门讨论归档存储引擎的论坛？112 号。

16.6 黑洞存储引擎

黑洞存储引擎充当一个“黑洞”，接受数据，但将其扔掉，不存储数据。检索始终返回空结果：

```
mysql>CREATE TABLE test(i INT, c CHAR(10))ENGINE=BLACKHOLE;
查询正常，0 行受影响(0.03 秒)
mysql>插入测试值(1, 'record one'),(2, 'record two');
查询正常，2 行受影响(0.00 秒)
记录:2 重复:0 警告:0
mysql>从测试中选择*;
空集(0.00 秒)
```

若要在从源代码构建 MySQL 时启用黑洞存储引擎，请使用-DWITH_BLACKHOLE_storage_engine 选项调用 CMake。

要检查黑洞引擎的源代码，请查看 MySQL 源代码发行版的 sql 目录。

创建黑洞表时，服务器将在全局数据字典中创建表定义。没有与表关联的文件。

黑洞存储引擎支持各种索引。也就是说，可以在表定义中包含索引声明。

黑洞存储引擎不支持分区。

您可以使用 SHOW ENGINES 语句检查黑洞存储引擎是否可用。

插入到黑洞表中不会存储任何数据，但如果启用了基于语句的二进制日志记录，则会记录 SQL 语句并将其复制到从属服务器。这可以用作中继器或过滤机制。

假设您的应用程序需要从端过滤规则，但是首先将所有二进制日志数据传输到从端会导致太多流量。在这种情况下，可以在主主机上设置一个默认存储引擎为黑洞的“虚拟”从进程，如下所示：

图 16.1 使用黑洞进行过滤的复制

主机写入其二进制日志。“dummy”mysqld 进程充当从进程，应用所需的 replicate do-*和 replicate ignore-*规则的组合，并编写自己的新的过滤二进制日志。请参阅第 17.1.6 节，“复制和二进制日志选项和变量”。此筛选日志提供给从服务器。

虚拟进程实际上并不存储任何数据，因此在复制主主机上运行额外的 mysqld 进程所产生的处理开销很小。这种类型的设置可以与其他复制从机一起重复。

黑洞表的 INSERT 触发器按预期工作。但是，由于黑洞表实际上并不存储任何数据，因此不会激活 UPDATE 和 DELETE 触发器：触发器定义中的 FOR EACH ROW 子句不适用，因为没有行。

黑洞存储引擎的其他可能用途包括：

•验证转储文件语法。

•通过比较启用和不启用二进制日志的黑洞性能，测量二进制日志的开销。
•BLACKHOLE 本质上是一个“无操作”存储引擎，因此它可用于查找与存储引擎本身无关的性能瓶颈。
黑洞引擎是事务感知的，因为提交的事务被写入二进制日志，而回滚的事务则不是。

黑洞引擎和自动增量列

黑洞引擎是一个不工作的引擎。在使用黑洞的工作台上执行的任何操作都将无效。在考虑自动递增的主键列的行为时，应该牢记这一点。引擎不会自动递增字段值，也不会保留自动递增字段状态。这对复制有重要影响。
考虑以下复制场景，其中应用以下三个条件：
1.在主服务器上有一个黑洞表，它有一个作为主键的自动递增字段。
2、在奴隶上存在同一个表，但使用 MyISAM 引擎。
3.在主表中执行插入操作时，不会在 INSERT 语句本身中显式设置自动增量值，也不会使用 SET INSERT\U ID 语句。
在这种情况下，复制将失败，在主键列上出现重复条目错误。
在基于语句的复制中，上下文事件中 INSERT_ID 的值将始终相同。因此，由于尝试插入主键列具有重复值的行，复制将失败。
在基于行的复制中，引擎为行返回的值对于每次插入总是相同的。这将导致从属服务器尝试使用主键列的相同值重播两个插入日志条目，因此复制将失败。

列筛选

使用基于行的复制时,(binlog_format=row)，支持从表中缺少最后一列的情况，如第 17.4.1.9 节“在主表和从表上使用不同表定义的复制”所述。
这种过滤在从属端工作，也就是说，列在被过滤掉之前被复制到从属端。至少有两种情况不希望将列复制到从属：
1.如果数据是机密的，那么从服务器不应该访问它。
2.如果主机有多个从机，在发送到从机之前进行过滤可能会减少网络流量。
使用黑洞引擎可以实现主柱过滤。这是以类似于如何实现主表过滤的方式执行的—使用黑洞引擎和--replicate do table 或--replicate ignore table 选项。
主机的设置为：

```
创建表 t1（public_col_1，…，public_col_N，
secret_col_1，…，secret_col_M）引擎=MyISAM；
```

受信任从属服务器的设置为：

```
创建表 t1（public_col_1，…，public_col_N）ENGINE=BLACKHOLE；
```

不受信任的从属服务器的设置为：

```
创建表 t1（public_col_1，…，public_col_N）ENGINE=MyISAM；
```

16.7 合并存储引擎

16.7.1 合并表的优缺点

16.7.2 合并表问题

合并存储引擎，也称为 MRG_MyISAM 引擎，是可以用作一个引擎的相同 MyISAM 表的集合。“相同”是指所有表具有相同的列数据类型和索引信息。不能合并列以不同顺序列出、相应列中的数据类型不完全相同或索引顺序不同的 MyISAM 表。但是，任何或所有 MyISAM 表都可以使用 myisampack 进行压缩。请参阅第 4.6.6 节,“myisampack-生成压缩的只读 MyISAM 表”。这些表格之间的差异无关紧要：

- 对应列和索引的名称可能不同。
- 表、列和索引的注释可能不同。
- 表选项（如 AVG_ROW_LENGTH、MAX_ROWS 或 PACK_KEYS）可能不同。

合并表的另一种选择是分区表，它将单个表的分区存储在单独的文件中，并使某些操作能够更有效地执行。有关更多信息，请参阅第 23 章，分区。

创建合并表时，MySQL 会在磁盘上创建一个.MRG 文件，其中包含应作为一个表使用的底层 MyISAM 表的名称。合并表的表格式存储在 MySQL 数据字典中。基础表不必与合并表位于同一数据库中。

可以在合并表上使用“选择”、“删除”、“更新”和“插入”。您必须对映射到合并表的 MyISAM 表具有选择、删除和更新权限。

注意

使用合并表会产生以下安全问题：如果用户有权访问 MyISAM 表 t，则该用户可以创建访问 t 的合并表 m。但是，如果用户对 t 的权限随后被吊销，则该用户可以通过 m 继续访问 t。对合并表使用 DROP TABLE 只删除合并规范。基础表不受影响。

要创建合并表，必须指定 UNION=（表列表）选项，该选项指示要使用哪个 MyISAM 表。您可以选择指定 INSERT_METHOD 选项来控制如何插入合并表。使用值 FIRST 或 LAST 分别在第一个或最后一个基础表中进行插入。

如果指定 no INSERT_METHOD 选项，或者使用 no 值指定该选项，则不允许插入合并表，并且尝试这样做会导致错误。

下面的示例演示如何创建合并表：

```
mysql>创建表 t1(
->一个 INT 不为空的自动递增主键,
->消息字符（20）引擎=MyISAM;
mysql>创建表 t2(
->一个 INT 不为空的自动递增主键,
->消息字符（20）引擎=MyISAM;
mysql>插入 t1（消息）值（'Testing'）,（'table'）,（'t1'）;
mysql>插入 t2（消息）值（'Testing'）,（'table'）,（'t2'）;
mysql>创建表总计(
->INT 不为空的自动递增,
->消息字符（20）, 索引（a）
->引擎=合并并集=（t1，t2）插入方法=最后一个;
```

列 a 作为主键在基础 MyISAM 表中编制索引，但不在合并表中编制索引。在那里它被索引，但不是作为主键，因为合并表不能对基础表集强制唯一性。（类似地，在基础表中具有唯一索引的列应在合并表中编制索引，但不能作为唯一索引。）

创建合并表后，可以使用它发出对整个表组进行操作的查询：

```
mysql>从 total 中选择*;
+---+---------+
```

|a |message|
+---+---------+
1	Testing
2	table
3	t1
1	Testing
2	table
3	t2
+---+---------+

要将合并表重新映射到不同的 MyISAM 表集合，可以使用以下方法之一：

•删除合并表并重新创建它。

•使用 ALTER TABLE tbl_name UNION=(…)更改基础表的列表。

也可以使用 ALTER TABLE。。。 UNION=(即，使用空 UNION 子句)删除所有基础表。

然而在这种情况下，表实际上是空的，插入操作失败，因为没有基础表接受新行。这样的表可以用作创建带有 CREATE table 的新合并表的模板。。。就像。

基础表定义和索引必须与合并表的定义紧密一致。打开作为合并表一部分的表时，而不是创建合并表时，将检查一致性。如果任何表未通过一致性检查，则触发表打开的操作将失败。这意味着当访问合并表时，对合并中表定义的更改可能会导致失败。应用于每个表的一致性检查包括：

•基础表和合并表的列数必须相同。

•基础表和合并表中的列顺序必须匹配。

•此外，还将比较父合并表和基础表中每个对应列的规范，并且必须满足这些检查：

o 基础表和合并表中的列类型必须相等。

o 基础表和合并表中的列长度必须相等。

o 基础表和合并表的列可以为空。

•基础表的索引数必须至少与合并表的索引数相同。基础表的索引可能比合并表多，但不能少。

注意

一个已知的问题存在于同一列中的索引必须在同一顺序中，合并表和底层的数据表中。见 Bug#33653。

每个索引必须满足这些检查：

o 基础表和合并表的索引类型必须相同。

o 基础表和合并表的索引定义中的索引部分(即复合索引中的多个列)数必须相同。

o 对于每个索引部分：

索引部件长度必须相等。

索引零件类型必须相等。

索引部件语言必须相等。

检查索引部分是否可以为空。

如果由于基础表出现问题而无法打开或使用合并表，则 CHECK table 将显示有关导致问题的表的信息。

额外资源

•在 https://forums.mysql.com/list.php 上有一个专门讨论合并存储引擎的论坛？93 年。

16.7.1 合并表的优缺点

合并表可以帮助您解决以下问题：
•轻松管理一组日志表。例如，您可以将不同月份的数据放在不同的表中，用 myisampack 压缩其中一些数据，然后创建一个合并表，将它们作为一个表使用。
•提高速度。您可以根据某些条件拆分大型只读表，然后将各个表放在不同的磁盘上。这样构造的合并表可能比使用单个大表快得多。
•执行更有效的搜索。如果您确切知道要查找的是什么，则可以只在一个基础表中搜索某些查询，并为其他查询使用合并表。甚至可以有许多不同的合并表使用重叠的表集。
•进行更有效的维修。修复映射到合并表的单个小表比修复单个大表更容易。
•立即将多张表映射为一张。合并表不需要维护自己的索引，因为它使用各个表的索引。因此，合并表集合的创建或重新映射速度非常快（创建合并表时，即使未创建索引，也必须指定索引定义）。
•如果您有一组按需创建大型表的表，则可以根据需要从中创建合并表。这样速度更快，节省了大量磁盘空间。
•超过操作系统的文件大小限制。每个 MyISAM 表都受此限制的约束，但 MyISAM 表的集合不受此限制。
•通过定义映射到 MyISAM 表的合并表，可以为 MyISAM 表创建别名或同义词。这样做应该不会对性能产生显著的影响（对于每个读操作，只有两个间接调用和 memcpy() 调用）。
合并表的缺点是：
•合并表只能使用相同的 MyISAM 表。
•某些 MyISAM 功能在合并表中不可用。例如，不能在合并表上创建全文索引（可以在基础 MyISAM 表上创建全文索引，但不能使用全文搜索搜索合并表）。
•如果合并表是非临时的，那么所有底层的 MyISAM 表都必须是非临时的。如果合并表是临时的，那么 MyISAM 表可以是临时表和非临时表的任意组合。
•合并表比 MyISAM 表使用更多的文件描述符。如果 10 个客户端使用映射到 10 个表的合并表，则服务器使用（10×10）+10 个文件描述符（10 个客户端的每个客户端有 10 个数据文件描述符，客户端之间共享 10 个索引文件描述符）。
•索引读取速度较慢。读取索引时，合并存储引擎需要对所有基础表发出读取操作，以检查哪个表与给定索引值最匹配。要读取下一个索引值，合并存储引擎需要搜索读取缓冲区以查找下一个值。只有当一个索引缓冲区用完时，存储引擎才需要读取下一个索引块。这使得合并索引在 eq-ref 搜索上慢得多，但在 ref 搜索上慢得多。有关 eq_ref 和 ref 的更多信息，请参阅第 13.8.2 节“解释语法”。

16.7.2 合并表问题

以下是合并表的已知问题：
•在 5.1.23 之前的 MySQL 服务器版本中，可以使用非临时子 MyISAM 表创建临时合并表。

在 5.1.23 版本中，合并子项通过父表锁定。如果父对象是临时的，则它不会被锁定，因此子对象也不会被锁定。并行使用 MyISAM 表损坏了它们。

•如果使用 ALTER TABLE 将合并表更改为另一个存储引擎，则将丢失到基础表的映射。相反，底层 MyISAM 表中的行被复制到修改后的表中，然后使用指定的存储引擎。

•合并表的 INSERT_METHOD table 选项指示要用于插入合并表的 MyISAM 表。但是，在将至少一行直接插入 MyISAM 表之前，对该 MyISAM 表使用 AUTO_INCREMENT table 选项对插入合并表没有影响。

•合并表不能在整个表上保持唯一性约束。执行插入时，数据将进入第一个或最后一个 MyISAM 表（由 INSERT_METHOD 选项确定）。

MYSQl 确保唯一的键值在 MyISAM 表中保持唯一，但不在集合中的所有基础表中保持唯一。

•由于合并引擎无法对基础表集强制唯一性，因此 REPLACE 无法按预期工作。两个关键事实是：

o REPLACE 只能在它要写入的基础表（由 INSERT_METHOD 选项决定）中检测唯一的密钥冲突。这与合并表本身的冲突不同。

o 如果 REPLACE 检测到一个唯一的密钥冲突，它将只更改它正在写入的基础表中的相应行；即第一个或最后一个表，由 INSERT_METHOD 选项确定。

类似的考虑适用于插入∞ 在重复密钥更新时。

•合并表不支持分区。也就是说，不能对合并表进行分区，也不能对合并表的任何底层 MyISAM 表进行分区。

•不应在映射到打开合并表的任何表上使用 ANALYZE TABLE、REPAIR TABLE、OPTIMIZE TABLE、ALTER TABLE、DROP TABLE、DELETE without a WHERE 子句或 TRUNCATE TABLE。如果这样做，合并表可能仍然引用原始表并产生意外结果。要解决此问题，请在执行任何命名操作之前发出 FLUSH tables 语句，确保没有合并表保持打开状态。

意外结果包括合并表上的操作可能会报告表损坏。如果这发生在底层 MyISAM 表上的一个命名操作之后，则损坏消息是假的。要处理这个问题，请在修改 MyISAM 表之后发出 FLUSH TABLES 语句。

•合并表正在使用的表上的 DROP TABLE 在 Windows 上不起作用，因为合并存储引擎的表映射在 MySQL 的上层是隐藏的。Windows 不允许删除打开的文件，因此在删除合并表之前，必须先刷新所有合并表（使用刷新表）或删除合并表。

•当访问 MyISAM 表和合并表时（例如，作为 SELECT 或 INSERT 语句的一部分），将检查这些表的定义。这些检查通过比较列顺序、类型、大小和关联索引，确保表定义与父合并表定义匹配。如果表之间存在差异，则返回错误，语句将失败。因为这些检查在打开表时进行，所以对单个表定义的任何更改，包括列更改，
列排序和引擎更改将导致语句失败。

•合并表及其基础表中的索引顺序应相同。如果使用 ALTER TABLE 向合并表中使用的表添加唯一索引，然后使用 ALTER TABLE 在合并表上添加非唯一索引，则如果基础表中已存在非唯一索引，则表的索引顺序将不同。（发生这种情况是因为 ALTER TABLE 将唯一索引放在非唯一索引之前，以便于快速检测重复键。）因此，对具有此类索引的表的查询可能会返回意外结果。

•如果遇到类似于错误 1017（HY000）的错误消息：找不到文件：“tbl_name.MRG”（errno:2），则通常表示某些基础表不使用 MyISAM 存储引擎。确认所有这些表都是 MyISAM。

合并表中的最大行数为 264（~1.844 e+1；与 MyISAM 表相同）。无法将多个 MyISAM 表合并

到一个合并表中，该合并表的行数将超过此数目。
•目前已知在父合并表中使用不同行格式的底层 MyISAM 表会失败。参见 Bug#32364。
•当锁表生效时，不能更改非临时合并表的联合列表。以下操作不起作用：
•创建表 m1。发动机=MRG_MYISAM。
•锁表 t1 写，t2 写，m1 写；
•更改表 m1。联合=（t1，t2）。
但是，可以使用临时合并表来执行此操作。
•不能使用 create 创建合并表。选择，既不作为临时合并表，也不作为非临时合并表。例如：
创建表 m1。引擎=MRG_MYISAM。选择。
尝试执行此操作会导致错误：tbl_name 不是基表。
•在某些情况下，如果基础表包含 CHAR 或 BINARY 列，则合并表和基础表之间的 PACK_KEYS 表选项值不同会导致意外结果。作为解决方法，使用 ALTER TABLE 确保所有涉及的表具有相同的 PACK_KEYS 值。（错误#50646）