

# MySQL 8.4官方文档阅读笔记

---

- `COUNT(*)` on a single table without a `WHERE` is retrieved directly from the table information for MyISAM and MEMORY tables. This is also done for any `NOT NULL` expression when used with only one table.

也就是说如果存储引擎是MyISAM或者MEMORY的话，表的数据行数会保存在表信息中（`SHOW TABLE STATUS LIKE '表名';`）。InnoDB因为支持事务的原因，则不会有这个信息。

- InnoDB Limits

- 一个表最多有1017个字段。
- 一个表最多有64的二级索引。
- 一个联合索引最多包含16个字段。
- 一行的数据最多不能超过页的一半，当有varchar类型的字段时，如果行的数据超过页的一半，该类型字段的数据会保存在溢出页上，在原行中保存溢出页的地址信息。BLOB、TEXT类型字段的数据，保存在溢出页（TINYBLOB、TINYTEXT除外）。行数据的总大小不能超过65535个字节（大BLOB、TEXT除外）。

- 临时表

- 文档目录地址：<https://dev.mysql.com/doc/refman/8.4/en/internal-temporary-tables.html>
- 在执行某些SQL查询（如 GROUP BY、DISTINCT、ORDER BY、UNION等）时，MySQL可能会创建临时表来存储中间计算结果，以便提高查询效率。  
explain时Extra列中如果出现Using temporary，则表示使用了临时表，`sql SHOW GLOBAL STATUS LIKE 'Created_tmp%tables';` Created\_tmp\_tables：创建的内存临时表数量。  
Created\_tmp\_disk\_tables：创建的磁盘临时表数量。
- 以下情况MySQL会使用磁盘临时表，而非内存临时表：
  - 当内存临时表的存储引擎为memory的时候，如果查询的字段中有TEXT或者BLOB数据类型。
  - 当执行select查询时，如果字段中包含字符串类型，其最大长度超过512，并且使用了UNION或UNION ALL，就会使用磁盘临时表。
  - 内存临时表大小超过限制。
- 配置表字段：
  - `internal_tmp_mem_storage_engine`：内存临时表使用的存储引擎，TempTable或者MEMORY，默认值是TempTable。
  - `tmp_table_size`：单个内存临时表最大大小，默认值是16M。超过这个值会退化为磁盘临时表。
  - `temptable_max_ram`：temptable存储引擎专属配置字段，所有内存临时表加起来的最大大小，超过这个值会退化为磁盘临时表。
  - `temptable_use_mmap`：temptable存储引擎专属配置字段，内存临时表超过`temptable_max_ram`的限制时，是否使用内存映射文件。默认是关闭。
  - `temptable_max_mmap`：temptable存储引擎专属配置字段，内存映射文件最大大小。
  - `max_heap_table_size`：当内存临时表的存储引擎为memory时，和`tmp_table_size`的最小值决定内存临时表的最大大小。

//TODO <https://dev.mysql.com/doc/refman/8.4/en/optimizing-innodb-diskio.html>

- Optimizing InnoDB Disk I/O
  - `innodb_flush_method` :
  - `innodb_use_native_aio` : 是否使用native aio异步IO，默认开启。文档中如是说：“With native AIO, the type of I/O scheduler has greater influence on I/O performance. Generally, noop and deadline I/O schedulers are recommended.”推荐挂载noop或者deadline IO Scheduler，但对于使用Nvme协议的SSD来说，应该不挂载任何IO Scheduler。
- InnoDB Buffer Pool
  - 文档目录地址：<https://dev.mysql.com/doc/refman/8.4/en/innodb-buffer-pool.html>
  - On dedicated servers, up to 80% of physical memory is often assigned to the buffer pool. 专属服务器上，推荐百分之八十以上内存当作buff pool。
  - 部分配置字段：
    - `innodb_buffer_pool_size` : Buff Pool大小，默认大小128M。If the server is started with `--innodb-dedicated-server`, the value of `innodb_buffer_pool_size` is set automatically if it is not explicitly defined。
    - `innodb_buffer_pool_chunk_size` : 每个chunk的大小，Buffer Pool是由若干chunk组成的。
    - `innodb_buffer_pool_instances` : 将InnoDB缓冲池划分为几个实例，每个instance有独立的缓存页链表和LRU。页号跟instance所对应，多个instance能够提升并发性能。过多也会造成内存浪费。
    - `innodb_old_blocks_pct` : 旧数据所占Buff Pool百分比，默认值37。
    - `innodb_old_blocks_time` : 旧数据移动到新数据最少等待时间。因为数据被加载到buff pool，并不一定是用户行为，有可能是mysql的预读机制导致。数据被加载到buff pool时，先放在old area,在`innodb_old_blocks_time`期间再被访问也不会移到new area，在`innodb_old_blocks_time`之后再次被访问，才会被移动到new area。
  - For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB. 推荐每个buffer pool instance最少1GB。
- Linear Read Ahead和Random Read Ahead
  - 线性预读和随机预读，InnoDB默认只使用线性预读机制。
  - 部分配置字段：
    - `innodb_read_ahead_threshold` : 取值范围0-64，这个单位是MySQL页的单位（MySQL的页默认大小为16K），一个extent有64个页。当一个extent已经被加载到缓存中的页的数量达到限制时，会触发线性预读，加载下一整个extent到buff pool中。  
PS：linux虚拟内存管理的基本单位是页，一般大小为4K。文件管理系统的基本单位是块，通常块和页的大小相等。扇区是磁盘的基本单位，块对应多个扇区。虽然SSD在物理设备中不存在扇区的概念，但为了兼容传统设备接口，在逻辑层依然维护扇区的抽象。
    - `innodb_random_read_ahead` : 是否开启随机预读，如果一个extent中的连续13个页都被加载到buff pool中，会触发随机预读，将这个extent的所有页都加载到buff pool中。默认是关闭的。
  - 全局变量：
    - `InnoDB_buffer_pool_read_ahead` : 通过预读机制加载到Buffer Pool中的数据页数量。
    - `InnoDB_buffer_pool_read_ahead_evicted` : 通过线性预读机制加载到Buffer Pool中，被访问前就被淘汰的数据页数量。

- `Innodb_buffer_pool_read_ahead_rnd`：通过随机预读机制加载到Buffer Pool中，被访问前就被淘汰的数据页数量。
- Buffer Pool刷盘
  - 部分配置字段：
    - `innodb_page_cleaners`：脏页清理线程，默认值是Buffer Pool实例的个数。逻辑扫描和准备脏页，实际IO操作由write io thread处理。
    - `innodb_read_io_threads`：read io thread数量。
    - `innodb_write_io_threads`：write io thread数量。
    - `innodb_max_dirty_pages_pct`：最大脏页百分比。默认值为90。
    - `innodb_max_dirty_pages_pct_lwm`：预刷新（pre-flushing）脏页百分比，默认值为10。
    - `innodb_flush_neighbors`：是否将同一个extent的其他脏页都刷新到磁盘。默认值是0，不刷新。值为1时，将同一extent的相邻脏页刷新到磁盘。值为2时，将同一extent的其他脏页都刷新到磁盘。
    - `innodb_lru_scan_depth`：每次扫描的深度，从LRU列表尾部开始，干净的就淘汰，脏页就假如列表等待刷盘。默认值为1024。
  - 自适应刷新：
    - 部分配置字段：
      - `innodb_adaptive_flushing`：是否开启自适应刷新。默认开启。
      - `innodb_adaptive_flushing_lwm`：百分比，当脏页数据达到这个百分比时，会强制启动自实现刷新。
  - Buff Pool刷盘步骤：当Buffer Pool实例中的脏页百分比超过`innodb_max_dirty_pages_pct_lwm`触发脏页刷盘，每次从LRU列表尾部查看`innodb_lru_scan_depth`数量的页，如果是脏页就放入一个脏页列表等待刷盘，如果是干净的就直接丢弃，释放buffer pool的空间，让脏页百分比低于`innodb_max_dirty_pages_pct`。对于脏页列表的刷盘速率，如果没有开启自适应刷新，且脏页数量没有达到`innodb_adaptive_flushing_lwm`限制时，以固定速率刷盘，否则自适应刷新启动，会根据当前IO压力等因素来决定刷盘速率。  
对于块设备是Nvme SSD并且开启了AIO时，如果不能打满SSD的单个处理队列时，`innodb_read_io_threads`和`innodb_write_io_threads`开启多个线程毫无意义，反而可能会增加额外的线程切换消耗。
- Change Buffer
  - 对非唯一索引进行insert、update、delete操作时，如果对应的数据不在buffer pool中时，不会立即从磁盘中加载数据到buffer pool中，而是将对应页的操作记录保存在change buffer。之后如果对应的页被加载到buffer pool时会进行合并操作。如果对应页一直未被加载到buffer pool，change buffer的数据在MySQL服务空闲时或者MySQL进程关闭时进行合并操作。Change Buffer在内存中会占用Buffer Pool的空间，磁盘上则是保存在系统表上。
  - 部分配置字段：
    - `innodb_change_buffering`：默认值none，不会缓存任何操作。  
none:默认值，不会缓存任何操作。  
inserts:缓存插入操作。  
deletes:缓存删除标记操作。  
changes:缓存插入和删除标记操作。  
purges:缓存真正的删除操作。  
all:缓存所有变更操作。

- `innodb_change_buffer_max_size` : Change Buffer所占Buffer Pool最大百分比。默认值25，最大值50。
- 8.4版本Change Buffer默认是关闭的，我觉得最主要的原因是，目前SSD已经非常普遍，SSD对于顺序IO和随机IO性能差异不大，反而Change Buffer会挤压Buffer Pool的空间，维护Change Buffer也会带来一定的性能消耗，所以默认关闭。但如果实际应用中如果存在大量使用非唯一索引进行更新时，使用Change Buffer可能是个更优的选择。
- Adaptive Hash Index
  - 部分配置字段：
    - `innodb_adaptive_hash_index` : 是否开启，默认关闭。
    - `innodb_adaptive_hash_index_parts` : 分成多少区，减少锁竞争，提升并发能力。
  - 只有使用非聚簇索引时，才会建立哈希项。如果某一页多次访问，被当成是热点数据，那么AHI会主动建立哈希项。对于同一个数据页的热点数据，多种访问路径会建立多个哈希项。AHI的应用会增加锁的消耗，并且当Buffer Pool中的数据页，被丢弃或者触发脏页回写时，为了维护AHI会增加额外的消耗。
- Doublewrite Buffer
  - 假设有一个脏页准备从Buffer Pool刷新到磁盘，首先会将脏页从Buffer Pool复制到Doublewrite Buffer，将Doublewrite Buffer中的数据刷盘后才会将Buffer Pool中的脏页刷盘。
  - 部分配置字段：
    - `innodb_doublewrite` : 是否开启Doublewrite Buffer。  
ON : 默认值开启  
OFF : 关闭  
DETECT\_AND\_RECOVER : DETECT\_AND\_RECOVER is the same as ON. 与ON相同  
DETECT\_ONLY : With DETECT\_ONLY, only metadata is written to the doublewrite buffer. 只保留元数据，InnoDB每个数据页开头都有一些结构化字段，比如：校验和、页编号等等。元数据应该指的就是这些数据，也就是说该选项只判断数据是否完成，不做数据恢复。
    - `innodb_doublewrite_dir`:doublewrite buffer磁盘文件路径。
    - `innodb_doublewrite_files`:每个innodb buffer pool实例的doublewrite buffer的磁盘文件数量。默认值是2。
    - `innodb_doublewrite_pages`:Defines the maximum number of doublewrite pages per thread for a batch write. 每个线程批量写入的最大页数。默认值128。那么一次写入的大小为128\*16K=2M。
- undo log
  - 部分配置字段：
    - `innodb_rollback_segments` : 回滚段的数量，默认值为128。
- redo log
  - redo log使用了page cache。redo log记录的是对数据页所做的修改操作的物理日志信息，比如“第X页第Y字节由a改为b”的二进制信息。
  - 部分配置字段：
    - `innodb_redo_log_capacity` : 所有redo log文件的磁盘空间大小。默认值为100M。如果该字段生效，redo log文件数量为32。

- `innodb_log_files_in_group`：redo log文件数量。默认值为2。
- `innodb_log_file_size`：单个redo log文件大小。默认值为48M。
- `innodb_log_buffer_size`：redo log内存缓存大小。默认值为64M。
- `innodb_log_write_ahead_size`：redo log写入磁盘的大小。默认值为8KB。
- `innodb_log_writer_threads`：是否开启一个专属的redo log写入线程。默认关闭。
- `innodb_flush_log_at_trx_commit`：redo log的刷盘策略。0：redo log每秒刷新到磁盘，事务提交只写入内存，不写入磁盘。1：每次事务提交，写入redo log buffer并立即刷盘到磁盘。2：每次事务提交，写入redo log buffer但不立即刷盘，每秒由后台线程统一刷盘。默认值为1。
- 如果定义了`innodb_redo_log_capacity`，那么`innodb_log_files_in_group`和`innodb_log_file_size`这两个变量将会被忽略，`innodb_redo_log_capacity`将取而代之。如果没有显式设置`innodb_redo_log_capacity`，但设置了`innodb_log_file_size`和`innodb_log_files_in_group`，那么redo log容量为 $\text{innodb\_log\_file\_size} * \text{innodb\_log\_files\_in\_group}$ 。
- binary log
  - binary log使用了page cache。
  - 部分配置字段：
    - `binlog_cache_size`：binlog cache buffer的大小，如果超过数据会存在临时文件。
    - `sync_binlog`：控制binary log何时刷盘。0：不主动刷盘，由操作系统自己控制。1：每次将user buffer中数据写入page cache后，主动刷盘。N：N个事务后主动刷盘。
    - `binlog_expire_logs_seconds`：binary log文件的过期时间。单位是秒。
    - `binlog_expire_logs_auto_purge`：binary log文件过期后是否自动删除，默认打开，如果关闭不会主动删除，需要手动删。
    - `binlog_format`：binary log格式。
      - STATEMENT：记录执行的SQL语句。
      - ROW：记录每一行数据的变化。
      - MIXED：默认使用STATEMENT，但在判断某条语句可能在statement-based复制下不安全时，会自动切换为ROW。
- 二阶段提交事务：当事务提交时，首先在redo log buffer中写入一条“prepare”日志，然后将redo log buffer write()到操作系统 page cache。接着将事务的binlog数据写入page cache。最后写入一条“commit”日志到redo log buffer，并同样write()到page cache。这两个日志的page cache脏页的实际刷盘时机由各自的配置控制：redo log由`innodb_flush_log_at_trx_commit`控制，binlog由`sync_binlog`控制。
- relay log