## 17.14 InnoDB Startup Options and System Variables

System variables that are true or false can be enabled at server startup by naming them, or disabled by using a `--skip-` prefix. For example, to enable or disable the `InnoDB` adaptive hash index, you can use `--innodb-adaptive-hash-index` or `--skip-innodb-adaptive-hash-index` on the command line, or `innodb_adaptive_hash_index` or `skip_innodb_adaptive_hash_index` in an option file.

Some variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. Boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case-sensitive), as well as `1` and `0`. See Section 6.2.2.4, “Program Option Modifiers”.

System variables that take a numeric value can be specified as `--var_name=value` on the command line or as `var_name=value` in option files.

Many system variables can be changed at runtime (see Section 7.1.9.2, “Dynamic System Variables”).

For information about `GLOBAL` and `SESSION` variable scope modifiers, refer to the `SET` statement documentation.

Certain options control the locations and layout of the `InnoDB` data files. Section 17.8.1, “InnoDB Startup Configuration” explains how to use these options.

Some options, which you might not use initially, help tune `InnoDB` performance characteristics based on machine capacity and database workload.

For more information on specifying options and system variables, see Section 6.2.2, “Specifying Program Options”.

**Table 17.23 InnoDB Option and Variable Reference**

| Name | Cmd-Line | Option File | System Var | Status Var | Var Scope | Dynamic |
|---|---|---|---|---|---|---|
| foreign_key_checks | | | Yes | | Both | Yes |
| innodb_adaptive_flushing | Yes | Yes | Yes | | Global | Yes |
| innodb_adaptive_flushing_lwm | Yes | Yes | Yes | | Global | Yes |
| innodb_adaptive_hash_index | Yes | Yes | Yes | | Global | Yes |
| innodb_adaptive_hash_index_parts | Yes | Yes | Yes | | Global | No |
| innodb_adaptive_max_sleep_delay | Yes | Yes | Yes | | Global | Yes |
| innodb_autoextend_increment | Yes | Yes | Yes | | Global | Yes |
| innodb_autoinc_lock_mode | Yes | Yes | Yes | | Global | No |
| innodb_background_drop_list_empty | Yes | Yes | Yes | | Global | Yes |
| Innodb_buffer_pool_bytes_data | | | | Yes | Global | No |
| Innodb_buffer_pool_bytes_dirty | | | | Yes | Global | No |
| innodb_buffer_pool_chunk_size | Yes | Yes | Yes | | Global | No |
| innodb_buffer_pool_debug | Yes | Yes | Yes | | Global | No |
| innodb_buffer_pool_dump_at_shutdown | Yes | Yes | Yes | | Global | Yes |
| innodb_buffer_pool_dump_now | Yes | Yes | Yes | | Global | Yes |
| innodb_buffer_pool_dump_pct | Yes | Yes | Yes | | Global | Yes |
| Innodb_buffer_pool_dump_status | | | | Yes | Global | No |
| innodb_buffer_pool_filename | Yes | Yes | Yes | | Global | Yes |
| innodb_buffer_pool_in_core_file | Yes | Yes | Yes | | Global | Yes |
| innodb_buffer_pool_instances | Yes | Yes | Yes | | Global | No |
| innodb_buffer_pool_load_abort | Yes | Yes | Yes | | Global | Yes |
| innodb_buffer_pool_load_at_startup | Yes | Yes | Yes | | Global | No |
| innodb_buffer_pool_load_now | Yes | Yes | Yes | | Global | Yes |
| Innodb_buffer_pool_load_status | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_data | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_dirty | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_flushed | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_free | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_latched | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_misc | | | | Yes | Global | No |
| Innodb_buffer_pool_pages_total | | | | Yes | Global | No |
| Innodb_buffer_pool_read_ahead | | | | Yes | Global | No |
| Innodb_buffer_pool_read_ahead_evicted | | | | Yes | Global | No |
| Innodb_buffer_pool_read_ahead_rnd | | | | Yes | Global | No |
| Innodb_buffer_pool_read_requests | | | | Yes | Global | No |
| Innodb_buffer_pool_reads | | | | Yes | Global | No |
| Innodb_buffer_pool_resize_status | | | | Yes | Global | No |

| | | | | | | |
|---|---|---|---|---|---|---|
| innodb_buffer_pool_size | Yes | Yes | Yes | | Global | Yes |
| Innodb_buffer_pool_wait_free | | | | Yes | Global | No |
| Innodb_buffer_pool_write_requests | | | | Yes | Global | No |
| innodb_change_buffer_max_size | Yes | Yes | Yes | | Global | Yes |
| innodb_change_buffering | Yes | Yes | Yes | | Global | Yes |
| innodb_change_buffering_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_checkpoint_disabled | Yes | Yes | Yes | | Global | Yes |
| innodb_checksum_algorithm | Yes | Yes | Yes | | Global | Yes |
| innodb_cmp_per_index_enabled | Yes | Yes | Yes | | Global | Yes |
| innodb_commit_concurrency | Yes | Yes | Yes | | Global | Yes |
| innodb_compress_debug | Yes | Yes | Yes | | Global | Yes |
| odb_compression_failure_threshold_pct | Yes | Yes | Yes | | Global | Yes |
| innodb_compression_level | Yes | Yes | Yes | | Global | Yes |
| innodb_compression_pad_pctmax | Yes | Yes | Yes | | Global | Yes |
| innodb_concurrency_tickets | Yes | Yes | Yes | | Global | Yes |
| innodb_data_file_path | Yes | Yes | Yes | | Global | No |
| Innodb_data_fsyncs | | | | Yes | Global | No |
| innodb_data_home_dir | Yes | Yes | Yes | | Global | No |
| Innodb_data_pending_fsyncs | | | | Yes | Global | No |
| Innodb_data_pending_reads | | | | Yes | Global | No |
| Innodb_data_pending_writes | | | | Yes | Global | No |
| Innodb_data_read | | | | Yes | Global | No |
| Innodb_data_reads | | | | Yes | Global | No |
| Innodb_data_writes | | | | Yes | Global | No |
| Innodb_data_written | | | | Yes | Global | No |
| Innodb_dblwr_pages_written | | | | Yes | Global | No |
| Innodb_dblwr_writes | | | | Yes | Global | No |
| innodb_ddl_buffer_size | Yes | Yes | Yes | | Session | Yes |
| innodb_ddl_log_crash_reset_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_ddl_threads | Yes | Yes | Yes | | Session | Yes |
| innodb_deadlock_detect | Yes | Yes | Yes | | Global | Yes |
| innodb_dedicated_server | Yes | Yes | Yes | | Global | No |
| innodb_default_row_format | Yes | Yes | Yes | | Global | Yes |
| innodb_directories | Yes | Yes | Yes | | Global | No |
| innodb_disable_sort_file_cache | Yes | Yes | Yes | | Global | Yes |
| innodb_doublewrite | Yes | Yes | Yes | | Global | Yes |
| innodb_doublewrite_batch_size | Yes | Yes | Yes | | Global | No |
| innodb_doublewrite_dir | Yes | Yes | Yes | | Global | No |
| innodb_doublewrite_files | Yes | Yes | Yes | | Global | No |
| innodb_doublewrite_pages | Yes | Yes | Yes | | Global | No |
| innodb_fast_shutdown | Yes | Yes | Yes | | Global | Yes |
| innodb_fil_make_page_dirty_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_file_per_table | Yes | Yes | Yes | | Global | Yes |
| innodb_fill_factor | Yes | Yes | Yes | | Global | Yes |
| innodb_flush_log_at_timeout | Yes | Yes | Yes | | Global | Yes |
| innodb_flush_log_at_trx_commit | Yes | Yes | Yes | | Global | Yes |
| innodb_flush_method | Yes | Yes | Yes | | Global | No |
| innodb_flush_neighbors | Yes | Yes | Yes | | Global | Yes |
| innodb_flush_sync | Yes | Yes | Yes | | Global | Yes |
| innodb_flushing_avg_loops | Yes | Yes | Yes | | Global | Yes |
| innodb_force_load_corrupted | Yes | Yes | Yes | | Global | No |
| innodb_force_recovery | Yes | Yes | Yes | | Global | No |
| innodb_fsync_threshold | Yes | Yes | Yes | | Global | Yes |
| innodb_ft_aux_table | | | Yes | | Global | Yes |
| innodb_ft_cache_size | Yes | Yes | Yes | | Global | No |
| innodb_ft_enable_diag_print | Yes | Yes | Yes | | Global | Yes |
| innodb_ft_enable_stopword | Yes | Yes | Yes | | Both | Yes |
| innodb_ft_max_token_size | Yes | Yes | Yes | | Global | No |
| innodb_ft_min_token_size | Yes | Yes | Yes | | Global | No |
| innodb_ft_num_word_optimize | Yes | Yes | Yes | | Global | Yes |
| innodb_ft_result_cache_limit | Yes | Yes | Yes | | Global | Yes |
| innodb_ft_server_stopword_table | Yes | Yes | Yes | | Global | Yes |
| innodb_ft_sort_pll_degree | Yes | Yes | Yes | | Global | No |
| innodb_ft_total_cache_size | Yes | Yes | Yes | | Global | No |
| innodb_ft_user_stopword_table | Yes | Yes | Yes | | Both | Yes |
| Innodb_have_atomic_builtins | | | | Yes | Global | No |
| innodb_idle_flush_pct | Yes | Yes | Yes | | Global | Yes |
| innodb_io_capacity | Yes | Yes | Yes | | Global | Yes |
| innodb_io_capacity_max | Yes | Yes | Yes | | Global | Yes |

| | | | | | | |
|---|---|---|---|---|---|---|
| innodb_limit_optimistic_insert_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_lock_wait_timeout | Yes | Yes | Yes | | Both | Yes |
| innodb_log_buffer_size | Yes | Yes | Yes | | Global | Yes |
| innodb_log_checkpoint_fuzzy_now | Yes | Yes | Yes | | Global | Yes |
| innodb_log_checkpoint_now | Yes | Yes | Yes | | Global | Yes |
| innodb_log_checksums | Yes | Yes | Yes | | Global | Yes |
| innodb_log_compressed_pages | Yes | Yes | Yes | | Global | Yes |
| innodb_log_file_size | Yes | Yes | Yes | | Global | No |
| innodb_log_files_in_group | Yes | Yes | Yes | | Global | No |
| innodb_log_group_home_dir | Yes | Yes | Yes | | Global | No |
| innodb_log_spin_cpu_abs_lwm | Yes | Yes | Yes | | Global | Yes |
| innodb_log_spin_cpu_pct_hwm | Yes | Yes | Yes | | Global | Yes |
| innodb_log_wait_for_flush_spin_hwm | Yes | Yes | Yes | | Global | Yes |
| Innodb_log_waits | | | | Yes | Global | No |
| innodb_log_write_ahead_size | Yes | Yes | Yes | | Global | Yes |
| Innodb_log_write_requests | | | | Yes | Global | No |
| innodb_log_writer_threads | Yes | Yes | Yes | | Global | Yes |
| Innodb_log_writes | | | | Yes | Global | No |
| innodb_lru_scan_depth | Yes | Yes | Yes | | Global | Yes |
| innodb_max_dirty_pages_pct | Yes | Yes | Yes | | Global | Yes |
| innodb_max_dirty_pages_pct_lwm | Yes | Yes | Yes | | Global | Yes |
| innodb_max_purge_lag | Yes | Yes | Yes | | Global | Yes |
| innodb_max_purge_lag_delay | Yes | Yes | Yes | | Global | Yes |
| innodb_max_undo_log_size | Yes | Yes | Yes | | Global | Yes |
| innodb_merge_threshold_set_all_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_monitor_disable | Yes | Yes | Yes | | Global | Yes |
| innodb_monitor_enable | Yes | Yes | Yes | | Global | Yes |
| innodb_monitor_reset | Yes | Yes | Yes | | Global | Yes |
| innodb_monitor_reset_all | Yes | Yes | Yes | | Global | Yes |
| Innodb_num_open_files | | | | Yes | Global | No |
| innodb_numa_interleave | Yes | Yes | Yes | | Global | No |
| innodb_old_blocks_pct | Yes | Yes | Yes | | Global | Yes |
| innodb_old_blocks_time | Yes | Yes | Yes | | Global | Yes |
| innodb_online_alter_log_max_size | Yes | Yes | Yes | | Global | Yes |
| innodb_open_files | Yes | Yes | Yes | | Global | Yes |
| innodb_optimize_fulltext_only | Yes | Yes | Yes | | Global | Yes |
| Innodb_os_log_fsyncs | | | | Yes | Global | No |
| Innodb_os_log_pending_fsyncs | | | | Yes | Global | No |
| Innodb_os_log_pending_writes | | | | Yes | Global | No |
| Innodb_os_log_written | | | | Yes | Global | No |
| innodb_page_cleaners | Yes | Yes | Yes | | Global | No |
| Innodb_page_size | | | | Yes | Global | No |
| innodb_page_size | Yes | Yes | Yes | | Global | No |
| Innodb_pages_created | | | | Yes | Global | No |
| Innodb_pages_read | | | | Yes | Global | No |
| Innodb_pages_written | | | | Yes | Global | No |
| innodb_parallel_read_threads | Yes | Yes | Yes | | Session | Yes |
| innodb_print_all_deadlocks | Yes | Yes | Yes | | Global | Yes |
| innodb_print_ddl_logs | Yes | Yes | Yes | | Global | Yes |
| innodb_purge_batch_size | Yes | Yes | Yes | | Global | Yes |
| innodb_purge_rseg_truncate_frequency | Yes | Yes | Yes | | Global | Yes |
| innodb_purge_threads | Yes | Yes | Yes | | Global | No |
| innodb_random_read_ahead | Yes | Yes | Yes | | Global | Yes |
| innodb_read_ahead_threshold | Yes | Yes | Yes | | Global | Yes |
| innodb_read_io_threads | Yes | Yes | Yes | | Global | No |
| innodb_read_only | Yes | Yes | Yes | | Global | No |
| innodb_redo_log_archive_dirs | Yes | Yes | Yes | | Global | Yes |
| innodb_redo_log_capacity | Yes | Yes | Yes | | Global | Yes |
| Innodb_redo_log_capacity_resized | | | | Yes | Global | No |
| Innodb_redo_log_checkpoint_lsn | | | | Yes | Global | No |
| Innodb_redo_log_current_lsn | | | | Yes | Global | No |
| Innodb_redo_log_enabled | | | | Yes | Global | No |
| innodb_redo_log_encrypt | Yes | Yes | Yes | | Global | Yes |
| Innodb_redo_log_flushed_to_disk_lsn | | | | Yes | Global | No |
| Innodb_redo_log_logical_size | | | | Yes | Global | No |
| Innodb_redo_log_physical_size | | | | Yes | Global | No |
| Innodb_redo_log_read_only | | | | Yes | Global | No |
| Innodb_redo_log_resize_status | | | | Yes | Global | No |
| Innodb_redo_log_uuid | | | | Yes | Global | No |

| | | | | | | |
|---|---|---|---|---|---|---|
| innodb_replication_delay | Yes | Yes | Yes | | Global | Yes |
| innodb_rollback_on_timeout | Yes | Yes | Yes | | Global | No |
| innodb_rollback_segments | Yes | Yes | Yes | | Global | Yes |
| Innodb_row_lock_current_waits | | | | Yes | Global | No |
| Innodb_row_lock_time | | | | Yes | Global | No |
| Innodb_row_lock_time_avg | | | | Yes | Global | No |
| Innodb_row_lock_time_max | | | | Yes | Global | No |
| Innodb_row_lock_waits | | | | Yes | Global | No |
| Innodb_rows_deleted | | | | Yes | Global | No |
| Innodb_rows_inserted | | | | Yes | Global | No |
| Innodb_rows_read | | | | Yes | Global | No |
| Innodb_rows_updated | | | | Yes | Global | No |
| innodb_saved_page_number_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_segment_reserve_factor | Yes | Yes | Yes | | Global | Yes |
| innodb_sort_buffer_size | Yes | Yes | Yes | | Global | No |
| innodb_spin_wait_delay | Yes | Yes | Yes | | Global | Yes |
| innodb_spin_wait_pause_multiplier | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_auto_recalc | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_include_delete_marked | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_method | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_on_metadata | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_persistent | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_persistent_sample_pages | Yes | Yes | Yes | | Global | Yes |
| innodb_stats_transient_sample_pages | Yes | Yes | Yes | | Global | Yes |
| innodb-status-file | Yes | Yes | | | | |
| innodb_status_output | Yes | Yes | Yes | | Global | Yes |
| innodb_status_output_locks | Yes | Yes | Yes | | Global | Yes |
| innodb_strict_mode | Yes | Yes | Yes | | Both | Yes |
| innodb_sync_array_size | Yes | Yes | Yes | | Global | No |
| innodb_sync_debug | Yes | Yes | Yes | | Global | No |
| innodb_sync_spin_loops | Yes | Yes | Yes | | Global | Yes |
| Innodb_system_rows_deleted | | | | Yes | Global | No |
| Innodb_system_rows_inserted | | | | Yes | Global | No |
| Innodb_system_rows_read | | | | Yes | Global | No |
| Innodb_system_rows_updated | | | | Yes | Global | No |
| innodb_table_locks | Yes | Yes | Yes | | Both | Yes |
| innodb_temp_data_file_path | Yes | Yes | Yes | | Global | No |
| innodb_temp_tablespaces_dir | Yes | Yes | Yes | | Global | No |
| innodb_thread_concurrency | Yes | Yes | Yes | | Global | Yes |
| innodb_thread_sleep_delay | Yes | Yes | Yes | | Global | Yes |
| innodb_tmpdir | Yes | Yes | Yes | | Both | Yes |
| Innodb_truncated_status_writes | | | | Yes | Global | No |
| innodb_trx_purge_view_update_only_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_trx_rseg_n_slots_debug | Yes | Yes | Yes | | Global | Yes |
| innodb_undo_directory | Yes | Yes | Yes | | Global | No |
| innodb_undo_log_encrypt | Yes | Yes | Yes | | Global | Yes |
| innodb_undo_log_truncate | Yes | Yes | Yes | | Global | Yes |
| innodb_undo_tablespaces | Yes | Yes | Yes | | Global | Yes |
| Innodb_undo_tablespaces_active | | | | Yes | Global | No |
| Innodb_undo_tablespaces_explicit | | | | Yes | Global | No |
| Innodb_undo_tablespaces_implicit | | | | Yes | Global | No |
| Innodb_undo_tablespaces_total | | | | Yes | Global | No |
| innodb_use_fdatasync | Yes | Yes | Yes | | Global | Yes |
| innodb_use_native_aio | Yes | Yes | Yes | | Global | No |
| innodb_validate_tablespace_paths | Yes | Yes | Yes | | Global | No |
| innodb_version | | | Yes | | Global | No |
| innodb_write_io_threads | Yes | Yes | Yes | | Global | No |
| unique_checks | | | Yes | | Both | Yes |

### InnoDB Command Options

`--innodb-status-file`

| | |
|---|---|
| **Command-Line Format** | `--innodb-status-file[={OFF\|ON}]` |
| **Type** | Boolean |
| **Default Value** | `OFF` |

The `--innodb-status-file` startup option controls whether `InnoDB` creates a file named `innodb_status.`*`pid`* in the data directory and writes `SHOW ENGINE INNODB STATUS` output to it every 15 seconds, approximately.

The `innodb_status.pid` file is not created by default. To create it, start **mysqld** with the `--innodb-status-file` option. `InnoDB` removes the file when the server is shut down normally. If an abnormal shutdown occurs, the status file may have to be removed manually.

The `--innodb-status-file` option is intended for temporary use, as `SHOW ENGINE INNODB STATUS` output generation can affect performance, and the `innodb_status.pid` file can become quite large over time.

For related information, see Section 17.17.2, “Enabling InnoDB Monitors”.

### InnoDB System Variables

`innodb_adaptive_flushing`

| | |
|---|---|
| **Command-Line Format** | `--innodb-adaptive-flushing[={OFF\|ON}]` |
| **System Variable** | `innodb_adaptive_flushing` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

Specifies whether to dynamically adjust the rate of flushing dirty pages in the `InnoDB` buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Section 17.8.3.5, “Configuring Buffer Pool Flushing” for more information. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_adaptive_flushing_lwm`

| | |
|---|---|
| **Command-Line Format** | `--innodb-adaptive-flushing-lwm=#` |
| **System Variable** | `innodb_adaptive_flushing_lwm` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `10` |
| **Minimum Value** | `0` |
| **Maximum Value** | `70` |

Defines the low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. For more information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”.

`innodb_adaptive_hash_index`

| | |
|---|---|
| **Command-Line Format** | `--innodb-adaptive-hash-index[={OFF\|ON}]` |
| **System Variable** | `innodb_adaptive_hash_index` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Whether the `InnoDB` adaptive hash index is enabled or disabled. It may be desirable, depending on your workload, to dynamically enable or disable adaptive hash indexing to improve query performance. Because the adaptive hash index may not be useful for all workloads, conduct benchmarks with it both enabled and disabled, using realistic workloads. See Section 17.5.3, “Adaptive Hash Index” for details.

This variable is disabled by default. You can modify this parameter using the `SET GLOBAL` statement, without restarting the server. Changing the setting at runtime requires privileges sufficient to set global system variables. See Section 7.1.9.1, “System Variable Privileges”. You can also use `--innodb-adaptive-hash-index` at server startup to enable it.

Disabling the adaptive hash index empties the hash table immediately. Normal operations can continue while the hash table is emptied, and executing queries that were using the hash table access the index B-trees directly instead. When the adaptive hash index is re-enabled, the hash table is populated again during normal operation.

Before MySQL 8.4, this option was enabled by default.

`innodb_adaptive_hash_index_parts`

| | |
|---|---|
| **Command-Line Format** | `--innodb-adaptive-hash-index-parts=#` |
| **System Variable** | `innodb_adaptive_hash_index_parts` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Numeric |
| **Default Value** | 8 |

| | |
|---|---|
| **Minimum Value** | 1 |
| **Maximum Value** | 512 |

Partitions the adaptive hash index search system. Each index is bound to a specific partition, with each partition protected by a separate latch.

The adaptive hash index search system is partitioned into 8 parts by default. The maximum setting is 512.

For related information, see Section 17.5.3, “Adaptive Hash Index”.

`innodb_adaptive_max_sleep_delay`

| | |
|---|---|
| **Command-Line Format** | `--innodb-adaptive-max-sleep-delay=#` |
| **System Variable** | `innodb_adaptive_max_sleep_delay` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `150000` |
| **Minimum Value** | `0` |
| **Maximum Value** | `1000000` |
| **Unit** | microseconds |

Permits `InnoDB` to automatically adjust the value of `innodb_thread_sleep_delay` up or down according to the current workload. Any nonzero value enables automated, dynamic adjustment of the `innodb_thread_sleep_delay` value, up to the maximum value specified in the `innodb_adaptive_max_sleep_delay` option. The value represents the number of microseconds. This option can be useful in busy systems, with greater than 16 `InnoDB` threads. (In practice, it is most valuable for MySQL systems with hundreds or thousands of simultaneous connections.)

For more information, see Section 17.8.4, “Configuring Thread Concurrency for InnoDB”.

`innodb_autoextend_increment`

| | |
|---|---|
| **Command-Line Format** | `--innodb-autoextend-increment=#` |
| **System Variable** | `innodb_autoextend_increment` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `64` |
| **Minimum Value** | `1` |
| **Maximum Value** | `1000` |
| **Unit** | megabytes |

The increment size (in megabytes) for extending the size of an auto-extending `InnoDB` system tablespace file when it becomes full. The default value is 64. For related information, see System Tablespace Data File Configuration, and Resizing the System Tablespace.

The `innodb_autoextend_increment` setting does not affect file-per-table tablespace files or general tablespace files. These files are auto-extending regardless of the `innodb_autoextend_increment` setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

`innodb_autoinc_lock_mode`

| | |
|---|---|
| **Command-Line Format** | `--innodb-autoinc-lock-mode=#` |
| **System Variable** | `innodb_autoinc_lock_mode` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 2 |
| | `0` |
| **Valid Values** | `1` |
| | `2` |

The lock mode to use for generating auto-increment values. Permissible values are 0, 1, or 2, for traditional, consecutive, or interleaved, respectively.

The default setting is 2 (interleaved), for compatibility with row-based replication.

For the characteristics of each lock mode, see InnoDB AUTO_INCREMENT Lock Modes.

`innodb_background_drop_list_empty`

| | |
|---|---|
| **Command-Line Format** | `--innodb-background-drop-list-empty[={OFF|ON}]` |

| | |
|---|---|
| **System Variable** | innodb_background_drop_list_empty |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Enabling the innodb_background_drop_list_empty debug option helps avoid test case failures by delaying table creation until the background drop list is empty. For example, if test case A places table t1 on the background drop list, test case B waits until the background drop list is empty before creating table t1.

innodb_buffer_pool_chunk_size

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-chunk-size=# |
| **System Variable** | innodb_buffer_pool_chunk_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 134217728 |
| **Minimum Value** | 1048576 |
| **Maximum Value** | innodb_buffer_pool_size / innodb_buffer_pool_instances |
| **Unit** | bytes |

innodb_buffer_pool_chunk_size defines the chunk size for InnoDB buffer pool resizing operations.

To avoid copying all buffer pool pages during resizing operations, the operation is performed in “chunks”. By default, innodb_buffer_pool_chunk_size is 128MB (134217728 bytes). The number of pages contained in a chunk depends on the value of innodb_page_size. innodb_buffer_pool_chunk_size can be increased or decreased in units of 1MB (1048576 bytes).

The following conditions apply when altering the innodb_buffer_pool_chunk_size value:

If innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances is larger than the current buffer pool size when the buffer pool is initialized, innodb_buffer_pool_chunk_size is truncated to innodb_buffer_pool_size / innodb_buffer_pool_instances.

Buffer pool size must always be equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances. If you alter innodb_buffer_pool_chunk_size, innodb_buffer_pool_size is automatically rounded to a value that is equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances. The adjustment occurs when the buffer pool is initialized.

> Important
>
> Care should be taken when changing innodb_buffer_pool_chunk_size, as changing this value can automatically increase the size of the buffer pool. Before changing innodb_buffer_pool_chunk_size, calculate its effect on innodb_buffer_pool_size to ensure that the resulting buffer pool size is acceptable.

To avoid potential performance issues, the number of chunks (innodb_buffer_pool_size / innodb_buffer_pool_chunk_size) should not exceed 1000.

The innodb_buffer_pool_size variable is dynamic, which permits resizing the buffer pool while the server is online. However, the buffer pool size must be equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances, and changing either of those variable settings requires restarting the server.

See Section 17.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

innodb_buffer_pool_debug

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-debug[={OFF\|ON}] |
| **System Variable** | innodb_buffer_pool_debug |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Enabling this option permits multiple buffer pool instances when the buffer pool is less than 1GB in size, ignoring the 1GB minimum buffer pool size constraint imposed on innodb_buffer_pool_instances. The innodb_buffer_pool_debug option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

innodb_buffer_pool_dump_at_shutdown

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-dump-at-shutdown[={OFF\|ON}] |
| **System Variable** | innodb_buffer_pool_dump_at_shutdown |
| **Scope** | Global |

| | |
|---|---|
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Specifies whether to record the pages cached in the InnoDB buffer pool when the MySQL server is shut down, to shorten the warmup process at the next restart. Typically used in combination with innodb_buffer_pool_load_at_startup. The innodb_buffer_pool_dump_pct option defines the percentage of most recently used buffer pool pages to dump.

Both innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup are enabled by default.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

innodb_buffer_pool_dump_now

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-dump-now[={OFF\|ON}] |
| **System Variable** | innodb_buffer_pool_dump_now |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Immediately makes a record of pages cached in the InnoDB buffer pool. Typically used in combination with innodb_buffer_pool_load_now.

Enabling innodb_buffer_pool_dump_now triggers the recording action but does not alter the variable setting, which always remains OFF or 0. To view buffer pool dump status after triggering a dump, query the Innodb_buffer_pool_dump_status variable.

Enabling innodb_buffer_pool_dump_now triggers the dump action but does not alter the variable setting, which always remains OFF or 0. To view buffer pool dump status after triggering a dump, query the Innodb_buffer_pool_dump_status variable.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

innodb_buffer_pool_dump_pct

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-dump-pct=# |
| **System Variable** | innodb_buffer_pool_dump_pct |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 25 |
| **Minimum Value** | 1 |
| **Maximum Value** | 100 |

Specifies the percentage of the most recently used pages for each buffer pool to read out and dump. The range is 1 to 100. The default value is 25. For example, if there are 4 buffer pools with 100 pages each, and innodb_buffer_pool_dump_pct is set to 25, the 25 most recently used pages from each buffer pool are dumped.

innodb_buffer_pool_filename

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-filename=file_name |
| **System Variable** | innodb_buffer_pool_filename |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | File name |
| **Default Value** | ib_buffer_pool |

Specifies the name of the file that holds the list of tablespace IDs and page IDs produced by innodb_buffer_pool_dump_at_shutdown or innodb_buffer_pool_dump_now. Tablespace IDs and page IDs are saved in the following format: space, page_id. By default, the file is named ib_buffer_pool and is located in the InnoDB data directory. A non-default location must be specified relative to the data directory.

A file name can be specified at runtime, using a SET statement:

```
SET GLOBAL innodb_buffer_pool_filename='file_name';
```

You can also specify a file name at startup, in a startup string or MySQL configuration file. When specifying a file name at startup, the file must exist or InnoDB returns a startup error indicating that there is no such file or directory.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

innodb_buffer_pool_in_core_file

| | |
|---|---|
| **Command-Line Format** | `--innodb-buffer-pool-in-core-file[={OFF\|ON}]` |
| **System Variable** | `innodb_buffer_pool_in_core_file` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Disabling (default) the `innodb_buffer_pool_in_core_file` variable reduces the size of core files by excluding `InnoDB` buffer pool pages.

To use this variable, the `core_file` variable must be enabled, and to disable this option the operating system must support the `MADV_DONTDUMP` non-POSIX extension to `madvise()`, which is supported in Linux 3.4 and later. For more information, see Section 17.8.3.7, “Excluding or Including Buffer Pool Pages from Core Files”.

This is disabled by default on systems that support `MADV_DONTDUMP`, which is typically only Linux and not macOS or Windows.

Before MySQL 8.4, this option was enabled by default.

`innodb_buffer_pool_instances`

| | |
|---|---|
| **Command-Line Format** | `--innodb-buffer-pool-instances=#` |
| **System Variable** | `innodb_buffer_pool_instances` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `see description` |
| **Minimum Value** | `1` |
| **Maximum Value** | `64` |

The number of regions that the `InnoDB` buffer pool is divided into. For systems with buffer pools in the multi-gigabyte range, dividing the buffer pool into separate instances can improve concurrency, by reducing contention as different threads read and write to cached pages. Each page that is stored in or read from the buffer pool is assigned to one of the buffer pool instances randomly, using a hashing function. Each buffer pool manages its own free lists, flush lists, LRUs, and all other data structures connected to a buffer pool, and is protected by its own buffer pool mutex.

The total buffer pool size is divided among all the buffer pools. 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.

If `innodb_buffer_pool_size` <= 1 GiB, then the default `innodb_buffer_pool_instances` value is 1.

If `innodb_buffer_pool_size` > 1 GiB, then the default `innodb_buffer_pool_instances` value is the minimum value from the following two calculated hints, within a range of 1-64:

Buffer pool hint: calculated as 1/2 of (`innodb_buffer_pool_size` / `innodb_buffer_pool_chunk_size`)

CPU hint: calculated as 1/4 of available logical processors

For related information, see Section 17.8.3.1, “Configuring InnoDB Buffer Pool Size”.

`innodb_buffer_pool_load_abort`

| | |
|---|---|
| **Command-Line Format** | `--innodb-buffer-pool-load-abort[={OFF\|ON}]` |
| **System Variable** | `innodb_buffer_pool_load_abort` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Interrupts the process of restoring `InnoDB` buffer pool contents triggered by `innodb_buffer_pool_load_at_startup` or `innodb_buffer_pool_load_now`.

Enabling `innodb_buffer_pool_load_abort` triggers the abort action but does not alter the variable setting, which always remains `OFF` or `0`. To view buffer pool load status after triggering an abort action, query the `Innodb_buffer_pool_load_status` variable.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

`innodb_buffer_pool_load_at_startup`

| | |
|---|---|
| **Command-Line Format** | `--innodb-buffer-pool-load-at-startup[={OFF\|ON}]` |
| **System Variable** | `innodb_buffer_pool_load_at_startup` |
| **Scope** | Global |

| | |
|---|---|
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Specifies that, on MySQL server startup, the InnoDB buffer pool is automatically warmed up by loading the same pages it held at an earlier time. Typically used in combination with innodb_buffer_pool_dump_at_shutdown.

Both innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup are enabled by default.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

innodb_buffer_pool_load_now

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-load-now[={OFF\|ON}] |
| **System Variable** | innodb_buffer_pool_load_now |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Immediately warms up the InnoDB buffer pool by loading data pages without waiting for a server restart. Can be useful to bring cache memory back to a known state during benchmarking or to ready the MySQL server to resume its normal workload after running queries for reports or maintenance.

Enabling innodb_buffer_pool_load_now triggers the load action but does not alter the variable setting, which always remains OFF or 0. To view buffer pool load progress after triggering a load, query the Innodb_buffer_pool_load_status variable.

For more information, see Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

innodb_buffer_pool_size

| | |
|---|---|
| **Command-Line Format** | --innodb-buffer-pool-size=# |
| **System Variable** | innodb_buffer_pool_size |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 134217728 |
| **Minimum Value** | 5242880 |
| **Maximum Value (64-bit platforms)** | 2**64-1 |
| **Maximum Value (32-bit platforms)** | 2**32-1 |
| **Unit** | bytes |

The size in bytes of the buffer pool, the memory area where InnoDB caches table and index data. The default value is 134217728 bytes (128MB). The maximum value depends on the CPU architecture; the maximum is 4294967295 ($2^{32}$-1) on 32-bit systems and 18446744073709551615 ($2^{64}$-1) on 64-bit systems. On 32-bit systems, the CPU architecture and operating system may impose a lower practical maximum size than the stated maximum. When the size of the buffer pool is greater than 1GB, setting innodb_buffer_pool_instances to a value greater than 1 can improve the scalability on a busy server.

A larger buffer pool requires less disk I/O to access the same table data more than once. On a dedicated database server, you might set the buffer pool size to 80% of the machine's physical memory size. Be aware of the following potential issues when configuring buffer pool size, and be prepared to scale back the size of the buffer pool if necessary.

- Competition for physical memory can cause paging in the operating system.
- InnoDB reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified buffer pool size.
- Address space for the buffer pool must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.
- The time to initialize the buffer pool is roughly proportional to its size. On instances with large buffer pools, initialization time might be significant. To reduce the initialization period, you can save the buffer pool state at server shutdown and restore it at server startup. See Section 17.8.3.6, “Saving and Restoring the Buffer Pool State”.

When you increase or decrease buffer pool size, the operation is performed in chunks. Chunk size is defined by the innodb_buffer_pool_chunk_size variable, which has a default of 128 MB.

Buffer pool size must always be equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances. If you alter the buffer pool size to a value that is not equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances, buffer pool size is automatically adjusted to a value that is equal to or a multiple of innodb_buffer_pool_chunk_size * innodb_buffer_pool_instances.

`innodb_buffer_pool_size` can be set dynamically, which allows you to resize the buffer pool without restarting the server. The `Innodb_buffer_pool_resize_status` status variable reports the status of online buffer pool resizing operations. See Section 17.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

If `innodb_dedicated_server` is enabled, the `innodb_buffer_pool_size` value is automatically configured if it is not explicitly defined. For more information, see Section 17.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

`innodb_change_buffer_max_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-change-buffer-max-size=#` |
| **System Variable** | `innodb_change_buffer_max_size` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `25` |
| **Minimum Value** | `0` |
| **Maximum Value** | `50` |

Maximum size for the `InnoDB` change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For more information, see Section 17.5.2, “Change Buffer”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_change_buffering`

| | |
|---|---|
| **Command-Line Format** | `--innodb-change-buffering=value` |
| **System Variable** | `innodb_change_buffering` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | `none` |
| **Valid Values** | `none`<br>`inserts`<br>`deletes`<br>`changes`<br>`purges`<br>`all` |

Whether `InnoDB` performs change buffering, an optimization that delays write operations to secondary indexes so that the I/O operations can be performed sequentially. Permitted values are described in the following table. Values may also be specified numerically.

**Table 17.24 Permitted Values for innodb_change_buffering**

| Value | Numeric Value | Description |
|---|---|---|
| **`none`** | `0` | Default. Do not buffer any operations. |
| **`inserts`** | `1` | Buffer insert operations. |
| **`deletes`** | `2` | Buffer delete marking operations; strictly speaking, the writes that mark index records for later deletion during a purge operation. |
| **`changes`** | `3` | Buffer inserts and delete-marking operations. |
| **`purges`** | `4` | Buffer the physical deletion operations that happen in the background. |
| **`all`** | `5` | Buffer inserts, delete-marking operations, and purges. |

Before MySQL 8.4, the default value was `all`.

For more information, see Section 17.5.2, “Change Buffer”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_change_buffering_debug`

| | |
|---|---|
| **Command-Line Format** | `--innodb-change-buffering-debug=#` |
| **System Variable** | `innodb_change_buffering_debug` |
| **Scope** | Global |
| **Dynamic** | Yes |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 2 |

Sets a debug flag for InnoDB change buffering. A value of 1 forces all changes to the change buffer. A value of 2 causes an unexpected exit at merge. A default value of 0 indicates that the change buffering debug flag is not set. This option is only available when debugging support is compiled in using the WITH_DEBUG **CMake** option.

innodb_checkpoint_disabled

| | |
|---|---|
| **Command-Line Format** | --innodb-checkpoint-disabled[={OFF\|ON}] |
| **System Variable** | innodb_checkpoint_disabled |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

This is a debug option that is only intended for expert debugging use. It disables checkpoints so that a deliberate server exit always initiates InnoDB recovery. It should only be enabled for a short interval, typically before running DML operations that write redo log entries that would require recovery following a server exit. This option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

innodb_checksum_algorithm

| | |
|---|---|
| **Command-Line Format** | --innodb-checksum-algorithm=value |
| **System Variable** | innodb_checksum_algorithm |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | crc32 |
| **Valid Values** | crc32<br>strict_crc32<br>innodb<br>strict_innodb<br>none<br>strict_none |

Specifies how to generate and verify the checksum stored in the disk blocks of InnoDB tablespaces. The default value for innodb_checksum_algorithm is crc32.

The value innodb is backward-compatible with earlier versions of MySQL. The value crc32 uses an algorithm that is faster to compute the checksum for every modified block, and to check the checksums for each disk read. It scans blocks 64 bits at a time, which is faster than the innodb checksum algorithm, which scans blocks 8 bits at a time. The value none writes a constant value in the checksum field rather than computing a value based on the block data. The blocks in a tablespace can use a mix of old, new, and no checksum values, being updated gradually as the data is modified; once blocks in a tablespace are modified to use the crc32 algorithm, the associated tables cannot be read by earlier versions of MySQL.

The strict form of a checksum algorithm reports an error if it encounters a valid but non-matching checksum value in a tablespace. It is recommended that you only use strict settings in a new instance, to set up tablespaces for the first time. Strict settings are somewhat faster, because they do not need to compute all checksum values during disk reads.

The following table shows the difference between the none, innodb, and crc32 option values, and their strict counterparts. none, innodb, and crc32 write the specified type of checksum value into each data block, but for compatibility accept other checksum values when verifying a block during a read operation. Strict settings also accept valid checksum values but print an error message when a valid non-matching checksum value is encountered. Using the strict form can make verification faster if all InnoDB data files in an instance are created under an identical innodb_checksum_algorithm value.

**Table 17.25 Permitted innodb_checksum_algorithm Values**

| Value | Generated checksum (when writing) | Permitted checksums (when reading) |
|---|---|---|
| **none** | A constant number. | Any of the checksums generated by none, innodb, or crc32. |
| **innodb** | A checksum calculated in software, using the original algorithm from InnoDB. | Any of the checksums generated by none, innodb, or crc32. |
| **crc32** | A checksum calculated using the crc32 algorithm, possibly done with a hardware assist. | Any of the checksums generated by none, innodb, or crc32. |

| | | |
|---|---|---|
| **strict_none** | A constant number | Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered. |
| **strict_innodb** | A checksum calculated in software, using the original algorithm from `InnoDB`. | Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered. |
| **strict_crc32** | A checksum calculated using the `crc32` algorithm, possibly done with a hardware assist. | Any of the checksums generated by `none`, `innodb`, or `crc32`. `InnoDB` prints an error message if a valid but non-matching checksum is encountered. |

- 

`innodb_cmp_per_index_enabled`

| | |
|---|---|
| **Command-Line Format** | `--innodb-cmp-per-index-enabled[={OFF\|ON}]` |
| **System Variable** | `innodb_cmp_per_index_enabled` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enables per-index compression-related statistics in the Information Schema `INNODB_CMP_PER_INDEX` table. Because these statistics can be expensive to gather, only enable this option on development, test, or replica instances during performance tuning related to `InnoDB` compressed tables.

For more information, see Section 28.4.8, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”, and Section 17.9.1.4, “Monitoring InnoDB Table Compression at Runtime”.

`innodb_commit_concurrency`

| | |
|---|---|
| **Command-Line Format** | `--innodb-commit-concurrency=#` |
| **System Variable** | `innodb_commit_concurrency` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `0` |
| **Minimum Value** | `0` |
| **Maximum Value** | `1000` |

The number of threads that can commit at the same time. A value of 0 (the default) permits any number of transactions to commit simultaneously.

The value of `innodb_commit_concurrency` cannot be changed at runtime from zero to nonzero or vice versa. The value can be changed from one nonzero value to another.

`innodb_compress_debug`

| | |
|---|---|
| **Command-Line Format** | `--innodb-compress-debug=value` |
| **System Variable** | `innodb_compress_debug` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | `none` |
| **Valid Values** | `none`<br>`zlib`<br>`lz4`<br>`lz4hc` |

Compresses all tables using a specified compression algorithm without having to define a `COMPRESSION` attribute for each table. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

For related information, see Section 17.9.2, “InnoDB Page Compression”.

`innodb_compression_failure_threshold_pct`

| | |
|---|---|
| **Command-Line Format** | `--innodb-compression-failure-threshold-pct=#` |

| | |
|---|---|
| **System Variable** | `innodb_compression_failure_threshold_pct` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `5` |
| **Minimum Value** | `0` |
| **Maximum Value** | `100` |

Defines the compression failure rate threshold for a table, as a percentage, at which point MySQL begins adding padding within compressed pages to avoid expensive compression failures. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified by `innodb_compression_pad_pct_max`. A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

For more information, see Section 17.9.1.6, “Compression for OLTP Workloads”.

`innodb_compression_level`

| | |
|---|---|
| **Command-Line Format** | `--innodb-compression-level=#` |
| **System Variable** | `innodb_compression_level` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `6` |
| **Minimum Value** | `0` |
| **Maximum Value** | `9` |

Specifies the level of zlib compression to use for `InnoDB` compressed tables and indexes. A higher value lets you fit more data onto a storage device, at the expense of more CPU overhead during compression. A lower value lets you reduce CPU overhead when storage space is not critical, or you expect the data is not especially compressible.

For more information, see Section 17.9.1.6, “Compression for OLTP Workloads”.

`innodb_compression_pad_pct_max`

| | |
|---|---|
| **Command-Line Format** | `--innodb-compression-pad-pct-max=#` |
| **System Variable** | `innodb_compression_pad_pct_max` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `50` |
| **Minimum Value** | `0` |
| **Maximum Value** | `75` |

Specifies the maximum percentage that can be reserved as free space within each compressed page, allowing room to reorganize the data and modification log within the page when a compressed table or index is updated and the data might be recompressed. Only applies when `innodb_compression_failure_threshold_pct` is set to a nonzero value, and the rate of compression failures passes the cutoff point.

For more information, see Section 17.9.1.6, “Compression for OLTP Workloads”.

`innodb_concurrency_tickets`

| | |
|---|---|
| **Command-Line Format** | `--innodb-concurrency-tickets=#` |
| **System Variable** | `innodb_concurrency_tickets` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `5000` |
| **Minimum Value** | `1` |
| **Maximum Value** | `4294967295` |

Determines the number of threads that can enter `InnoDB` concurrently. A thread is placed in a queue when it tries to enter `InnoDB` if the number of threads has already reached the concurrency limit. When a thread is permitted to enter `InnoDB`, it is given a number of “ tickets” equal to the value of `innodb_concurrency_tickets`, and the thread can enter and leave `InnoDB` freely until it has used up its tickets. After that point, the thread again becomes subject to the concurrency check (and possible queuing) the next time it tries to enter `InnoDB`. The default value is 5000.

With a small `innodb_concurrency_tickets` value, small transactions that only need to process a few rows compete fairly with larger transactions that process many rows. The disadvantage of a small `innodb_concurrency_tickets` value is that large transactions must loop through the queue many times before they can complete, which extends the amount of time required to complete their task.

With a large innodb_concurrency_tickets value, large transactions spend less time waiting for a position at the end of the queue (controlled by innodb_thread_concurrency) and more time retrieving rows. Large transactions also require fewer trips through the queue to complete their task. The disadvantage of a large innodb_concurrency_tickets value is that too many large transactions running at the same time can starve smaller transactions by making them wait a longer time before executing.

With a nonzero innodb_thread_concurrency value, you may need to adjust the innodb_concurrency_tickets value up or down to find the optimal balance between larger and smaller transactions. The SHOW ENGINE INNODB STATUS report shows the number of tickets remaining for an executing transaction in its current pass through the queue. This data may also be obtained from the TRX_CONCURRENCY_TICKETS column of the Information Schema INNODB_TRX table.

For more information, see Section 17.8.4, “Configuring Thread Concurrency for InnoDB”.

innodb_data_file_path

| | |
|---|---|
| **Command-Line Format** | --innodb-data-file-path=file_name |
| **System Variable** | innodb_data_file_path |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | String |
| **Default Value** | ibdata1:12M:autoextend |

Defines the name, size, and attributes of InnoDB system tablespace data files. If you do not specify a value for innodb_data_file_path, the default behavior is to create a single auto-extending data file, slightly larger than 12MB, named ibdata1.

The full syntax for a data file specification includes the file name, file size, autoextend attribute, and max attribute:

*file_name*:*file_size*[:autoextend[:max:*max_file_size*]]

File sizes are specified in kilobytes, megabytes, or gigabytes by appending K, M or G to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, KB values are rounded to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

For additional configuration information, see System Tablespace Data File Configuration. For resizing instructions, see Resizing the System Tablespace.

innodb_data_home_dir

| | |
|---|---|
| **Command-Line Format** | --innodb-data-home-dir=dir_name |
| **System Variable** | innodb_data_home_dir |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |

The common part of the directory path for InnoDB system tablespace data files. The default value is the MySQL data directory. The setting is concatenated with the innodb_data_file_path setting, unless that setting is defined with an absolute path.

A trailing slash is required when specifying a value for innodb_data_home_dir. For example:

```
[mysqld]
innodb_data_home_dir = /path/to/myibdata/
```

This setting does not affect the location of file-per-table tablespaces.

For related information, see Section 17.8.1, “InnoDB Startup Configuration”.

innodb_ddl_buffer_size

| | |
|---|---|
| **Command-Line Format** | --innodb-ddl-buffer-size=# |
| **System Variable** | innodb_ddl_buffer_size |
| **Scope** | Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 1048576 |
| **Minimum Value** | 65536 |
| **Maximum Value** | 4294967295 |
| **Unit** | bytes |

Defines the maximum buffer size for DDL operations. The default setting is 1048576 bytes (approximately 1 MB). Applies to online DDL operations that create or rebuild secondary indexes. See Section 17.12.4, “Online DDL Memory Management”. The maximum buffer size per DDL thread is the maximum buffer size divided by the number of DDL threads (innodb_ddl_buffer_size/innodb_ddl_threads).

`innodb_ddl_log_crash_reset_debug`

| | |
|---|---|
| **Command-Line Format** | `--innodb-ddl-log-crash-reset-debug[={OFF\|ON}]` |
| **System Variable** | `innodb_ddl_log_crash_reset_debug` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enable this debug option to reset DDL log crash injection counters to 1. This option is only available when debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

`innodb_ddl_threads`

| | |
|---|---|
| **Command-Line Format** | `--innodb-ddl-threads=#` |
| **System Variable** | `innodb_ddl_threads` |
| **Scope** | Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 4 |
| **Minimum Value** | 1 |
| **Maximum Value** | 64 |

Defines the maximum number of parallel threads for the sort and build phases of index creation. Applies to online DDL operations that create or rebuild secondary indexes. For related information, see Section 17.12.5, “Configuring Parallel Threads for Online DDL Operations”, and Section 17.12.4, “Online DDL Memory Management”.

`innodb_deadlock_detect`

| | |
|---|---|
| **Command-Line Format** | `--innodb-deadlock-detect[={OFF\|ON}]` |
| **System Variable** | `innodb_deadlock_detect` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

This option is used to disable deadlock detection. On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the `innodb_lock_wait_timeout` setting for transaction rollback when a deadlock occurs.

For related information, see Section 17.7.5.2, “Deadlock Detection”.

`innodb_dedicated_server`

| | |
|---|---|
| **Command-Line Format** | `--innodb-dedicated-server[={OFF\|ON}]` |
| **System Variable** | `innodb_dedicated_server` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

When `innodb_dedicated_server` is enabled, `InnoDB` automatically configures the following variables:

- `innodb_buffer_pool_size`
- `innodb_redo_log_capacity`.

  Note

  `innodb_log_file_size` and `innodb_log_files_in_group` are deprecated, and superseded by `innodb_redo_log_capacity`; see Section 17.6.5, “Redo Log”.

- Before MySQL 8.4, `innodb_flush_method` was also automatically configured.

Only consider enabling `innodb_dedicated_server` if the MySQL instance resides on a dedicated server where it can use all available system resources. Enabling `innodb_dedicated_server` is not recommended if the MySQL instance shares system resources with other applications.

For more information, see Section 17.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

innodb_default_row_format

| | |
|---|---|
| **Command-Line Format** | `--innodb-default-row-format=value` |
| **System Variable** | innodb_default_row_format |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | `DYNAMIC`<br>`REDUNDANT` |
| **Valid Values** | `COMPACT`<br>`DYNAMIC` |

The innodb_default_row_format option defines the default row format for `InnoDB` tables and user-created temporary tables. The default setting is `DYNAMIC`. Other permitted values are `COMPACT` and `REDUNDANT`. The `COMPRESSED` row format, which is not supported for use in the system tablespace, cannot be defined as the default.

Newly created tables use the row format defined by innodb_default_row_format when a `ROW_FORMAT` option is not specified explicitly or when `ROW_FORMAT=DEFAULT` is used.

When a `ROW_FORMAT` option is not specified explicitly or when `ROW_FORMAT=DEFAULT` is used, any operation that rebuilds a table also silently changes the row format of the table to the format defined by innodb_default_row_format. For more information, see Defining the Row Format of a Table.

Internal `InnoDB` temporary tables created by the server to process queries use the `DYNAMIC` row format, regardless of the innodb_default_row_format setting.

innodb_directories

| | |
|---|---|
| **Command-Line Format** | `--innodb-directories=dir_name` |
| **System Variable** | innodb_directories |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |
| **Default Value** | `NULL` |

Defines directories to scan at startup for tablespace files. This option is used when moving or restoring tablespace files to a new location while the server is offline. It is also used to specify directories of tablespace files created using an absolute path or that reside outside of the data directory.

Tablespace discovery during crash recovery relies on the innodb_directories setting to identify tablespaces referenced in the redo logs. For more information, see Tablespace Discovery During Crash Recovery.

The default value is NULL, but directories defined by innodb_data_home_dir, innodb_undo_directory, and datadir are always appended to the innodb_directories argument value when `InnoDB` builds a list of directories to scan at startup. These directories are appended regardless of whether an innodb_directories setting is specified explicitly.

innodb_directories may be specified as an option in a startup command or in a MySQL option file. Quotes surround the argument value because otherwise some command interpreters interpret semicolon (`;`) as a special character. (For example, Unix shells treat it as a command terminator.)

Startup command:

```
mysqld --innodb-directories="directory_path_1;directory_path_2"
```

MySQL option file:

```
[mysqld]
innodb_directories="directory_path_1;directory_path_2"
```

Wildcard expressions cannot be used to specify directories.

The innodb_directories scan also traverses the subdirectories of specified directories. Duplicate directories and subdirectories are discarded from the list of directories to be scanned.

For more information, see Section 17.6.3.6, “Moving Tablespace Files While the Server is Offline”.

innodb_disable_sort_file_cache

| | |
|---|---|
| **Command-Line Format** | `--innodb-disable-sort-file-cache[={OFF\|ON}]` |
| **System Variable** | innodb_disable_sort_file_cache |
| **Scope** | Global |
| **Dynamic** | Yes |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Disables the operating system file system cache for merge-sort temporary files. The effect is to open such files with the equivalent of O_DIRECT.

innodb_doublewrite

| | |
|---|---|
| **Command-Line Format** | --innodb-doublewrite=value |
| **System Variable** | innodb_doublewrite |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | ON |
| **Valid Values** | ON<br>OFF<br>DETECT_AND_RECOVER<br>DETECT_ONLY |

The innodb_doublewrite variable controls doublewrite buffering. Doublewrite buffering is enabled by default in most cases.

You can set innodb_doublewrite to ON or OFF when starting the server to enable or disable doublewrite buffering, respectively. DETECT_AND_RECOVER is the same as ON. With this setting, except that the doublewrite buffer is fully enabled, with database page content written to the doublewrite buffer where it is accessed during recovery to fix incomplete page writes. With DETECT_ONLY, only metadata is written to the doublewrite buffer. Database page content is not written to the doublewrite buffer, and recovery does not use the doublewrite buffer to fix incomplete page writes. This lightweight setting is intended for detecting incomplete page writes only.

MySQL supports dynamic changes to the innodb_doublewrite setting that enables the doublewrite buffer, between ON, DETECT_AND_RECOVER, and DETECT_ONLY. MySQL does not support dynamic changes between a setting that enables the doublewrite buffer and OFF or vice versa.

If the doublewrite buffer is located on a Fusion-io device that supports atomic writes, the doublewrite buffer is automatically disabled and data file writes are performed using Fusion-io atomic writes instead. However, be aware that the innodb_doublewrite setting is global. When the doublewrite buffer is disabled, it is disabled for all data files including those that do not reside on Fusion-io hardware. This feature is only supported on Fusion-io hardware and is only enabled for Fusion-io NVMFS on Linux. To take full advantage of this feature, an innodb_flush_method setting of O_DIRECT is recommended.

For related information, see Section 17.6.4, “Doublewrite Buffer”.

innodb_doublewrite_batch_size

| | |
|---|---|
| **Command-Line Format** | --innodb-doublewrite-batch-size=# |
| **System Variable** | innodb_doublewrite_batch_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 256 |

Defines the number of doublewrite pages to write in a batch.

For more information, see Section 17.6.4, “Doublewrite Buffer”.

innodb_doublewrite_dir

| | |
|---|---|
| **Command-Line Format** | --innodb-doublewrite-dir=dir_name |
| **System Variable** | innodb_doublewrite_dir |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |

Defines the directory for doublewrite files. If no directory is specified, doublewrite files are created in the innodb_data_home_dir directory, which defaults to the data directory if unspecified.

For more information, see Section 17.6.4, “Doublewrite Buffer”.

innodb_doublewrite_files

| | |
|---|---|
| **Command-Line Format** | --innodb-doublewrite-files=# |

| | |
|---|---|
| **System Variable** | `innodb_doublewrite_files` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 2 |
| **Minimum Value** | 1 |
| **Maximum Value** | 256 |

Defines the number of doublewrite files. By default, two doublewrite files are created for each buffer pool instance.

At a minimum, there are two doublewrite files. The maximum number of doublewrite files is two times the number of buffer pool instances. (The number of buffer pool instances is controlled by the `innodb_buffer_pool_instances` variable.)

For more information, see Section 17.6.4, “Doublewrite Buffer”.

`innodb_doublewrite_pages`

| | |
|---|---|
| **Command-Line Format** | `--innodb-doublewrite-pages=#` |
| **System Variable** | `innodb_doublewrite_pages` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 128 |
| **Minimum Value** | 1 |
| **Maximum Value** | 512 |

Defines the maximum number of doublewrite pages per thread for a batch write. If no value is specified, `innodb_doublewrite_pages` defaults to 128.

Before MySQL 8.4, the default value was the `innodb_write_io_threads` value, which is 4 by default.

For more information, see Section 17.6.4, “Doublewrite Buffer”.

`innodb_extend_and_initialize`

| | |
|---|---|
| **Command-Line Format** | `--innodb=extend-and-initialize[={OFF\|ON}]` |
| **System Variable** | `innodb_extend_and_initialize` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

Controls how space is allocated to file-per-table and general tablespaces on Linux systems.

When enabled, `InnoDB` writes NULLs to newly allocated pages. When disabled, space is allocated using `posix_fallocate()` calls, which reserve space without physically writing NULLs.

For more information, see Section 17.6.3.8, “Optimizing Tablespace Space Allocation on Linux”.

`innodb_fast_shutdown`

| | |
|---|---|
| **Command-Line Format** | `--innodb-fast-shutdown=#` |
| **System Variable** | `innodb_fast_shutdown` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 1 |
| | 0 |
| **Valid Values** | 1 |
| | 2 |

The `InnoDB` shutdown mode. If the value is 0, `InnoDB` does a slow shutdown, a full purge and a change buffer merge before shutting down. If the value is 1 (the default), `InnoDB` skips these operations at shutdown, a process known as a fast shutdown. If the value is 2, `InnoDB` flushes its logs and shuts down cold, as if MySQL had crashed; no committed transactions are lost, but the crash recovery operation makes the next startup take longer.

The slow shutdown can take minutes, or even hours in extreme cases where substantial amounts of data are still buffered. Use the slow shutdown technique before upgrading or downgrading between MySQL major releases, so that all data files are fully prepared in case the upgrade process updates the file format.

Use innodb_fast_shutdown=2 in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

innodb_fil_make_page_dirty_debug

| | |
|---|---|
| **Command-Line Format** | --innodb-fil-make-page-dirty-debug=# |
| **System Variable** | innodb_fil_make_page_dirty_debug |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 2**32-1 |

By default, setting innodb_fil_make_page_dirty_debug to the ID of a tablespace immediately dirties the first page of the tablespace. If innodb_saved_page_number_debug is set to a non-default value, setting innodb_fil_make_page_dirty_debug dirties the specified page. The innodb_fil_make_page_dirty_debug option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

innodb_file_per_table

| | |
|---|---|
| **Command-Line Format** | --innodb-file-per-table[={OFF\|ON}] |
| **System Variable** | innodb_file_per_table |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

When innodb_file_per_table is enabled, tables are created in file-per-table tablespaces by default. When disabled, tables are created in the system tablespace by default. For information about file-per-table tablespaces, see Section 17.6.3.2, “File-Per-Table Tablespaces”. For information about the InnoDB system tablespace, see Section 17.6.3.1, “The System Tablespace”.

The innodb_file_per_table variable can be configured at runtime using a SET GLOBAL statement, specified on the command line at startup, or specified in an option file. Configuration at runtime requires privileges sufficient to set global system variables (see Section 7.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

When a table that resides in a file-per-table tablespace is truncated or dropped, the freed space is returned to the operating system. Truncating or dropping a table that resides in the system tablespace only frees space in the system tablespace. Freed space in the system tablespace can be used again for InnoDB data but is not returned to the operating system, as system tablespace data files never shrink.

The innodb_file_per-table setting does not affect the creation of temporary tables; temporary tables are created in session temporary tablespaces. See Section 17.6.3.5, “Temporary Tablespaces”.

innodb_fill_factor

| | |
|---|---|
| **Command-Line Format** | --innodb-fill-factor=# |
| **System Variable** | innodb_fill_factor |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 100 |
| **Minimum Value** | 10 |
| **Maximum Value** | 100 |

InnoDB performs a bulk load when creating or rebuilding indexes. This method of index creation is known as a “sorted index build”.

innodb_fill_factor defines the percentage of space on each B-tree page that is filled during a sorted index build, with the remaining space reserved for future index growth. For example, setting innodb_fill_factor to 80 reserves 20 percent of the space on each B-tree page for future index growth. Actual percentages may vary. The innodb_fill_factor setting is interpreted as a hint rather than a hard limit.

An innodb_fill_factor setting of 100 leaves 1/16 of the space in clustered index pages free for future index growth.

innodb_fill_factor applies to both B-tree leaf and non-leaf pages. It does not apply to external pages used for TEXT or BLOB entries.

For more information, see Section 17.6.2.3, “Sorted Index Builds”.

innodb_flush_log_at_timeout

| | |
|---|---|
| **Command-Line Format** | --innodb-flush-log-at-timeout=# |
| **System Variable** | innodb_flush_log_at_timeout |
| **Scope** | Global |
| **Dynamic** | Yes |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 1 |
| **Minimum Value** | 1 |
| **Maximum Value** | 2700 |
| **Unit** | seconds |

Write and flush the logs every *N* seconds. `innodb_flush_log_at_timeout` allows the timeout period between flushes to be increased in order to reduce flushing and avoid impacting performance of binary log group commit. The default setting for `innodb_flush_log_at_timeout` is once per second.

`innodb_flush_log_at_trx_commit`

| | |
|---|---|
| **Command-Line Format** | `--innodb-flush-log-at-trx-commit=#` |
| **System Variable** | `innodb_flush_log_at_trx_commit` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | 1<br>0 |
| **Valid Values** | 1<br>2 |

Controls the balance between strict ACID compliance for commit operations and higher performance that is possible when commit-related I/O operations are rearranged and done in batches. You can achieve better performance by changing the default value but then you can lose transactions in a crash.

The default setting of 1 is required for full ACID compliance. Logs are written and flushed to disk at each transaction commit.

With a setting of 0, logs are written and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.

With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.

For settings 0 and 2, once-per-second flushing is not 100% guaranteed. Flushing may occur more frequently due to DDL changes and other internal `InnoDB` activities that cause logs to be flushed independently of the `innodb_flush_log_at_trx_commit` setting, and sometimes less frequently due to scheduling issues. If logs are flushed once per second, up to one second of transactions can be lost in a crash. If logs are flushed more or less frequently than once per second, the amount of transactions that can be lost varies accordingly.

Log flushing frequency is controlled by `innodb_flush_log_at_timeout`, which allows you to set log flushing frequency to *N* seconds (where *N* is `1 ... 2700`, with a default value of 1). However, any unexpected **mysqld** process exit can erase up to *N* seconds of transactions.

DDL changes and other internal `InnoDB` activities flush the log independently of the `innodb_flush_log_at_trx_commit` setting.

`InnoDB` crash recovery works regardless of the `innodb_flush_log_at_trx_commit` setting. Transactions are either applied entirely or erased entirely.

For durability and consistency in a replication setup that uses `InnoDB` with transactions:

If binary logging is enabled, set `sync_binlog=1`.

Always set `innodb_flush_log_at_trx_commit=1`.

For information on the combination of settings on a replica that is most resilient to unexpected halts, see Section 19.4.2, “Handling an Unexpected Halt of a Replica”.

> Caution
>
> Many operating systems and some disk hardware fool the flush-to-disk operation. They may tell **mysqld** that the flush has taken place, even though it has not. In this case, the durability of transactions is not guaranteed even with the recommended settings, and in the worst case, a power outage can corrupt `InnoDB` data. Using a battery-backed disk cache in the SCSI disk controller or in the disk itself speeds up file flushes, and makes the operation safer. You can also try to disable the caching of disk writes in hardware caches.

`innodb_flush_method`

| | |
|---|---|
| **Command-Line Format** | `--innodb-flush-method=value` |
| **System Variable** | `innodb_flush_method` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | String |

| | |
|---|---|
| **Default Value (Unix)** | `O_DIRECT if supported, otherwise fsync` |
| **Default Value (Windows)** | `unbuffered` |
| **Valid Values (Unix)** | `fsync`<br>`O_DSYNC`<br>`littlesync`<br>`nosync`<br>`O_DIRECT`<br>`O_DIRECT_NO_FSYNC` |
| **Valid Values (Windows)** | `unbuffered`<br>`normal` |

Defines the method used to flush data to `InnoDB` data files and log files, which can affect I/O throughput.

On Unix-like systems, the default value is `O_DIRECT` if supported otherwise defaults to `fsync`. On Windows, the default value is `unbuffered`.

The `innodb_flush_method` options for Unix-like systems include:

- `fsync` or `0`: `InnoDB` uses the `fsync()` system call to flush both the data and log files.
- `O_DSYNC` or `1`: `InnoDB` uses `O_SYNC` to open and flush the log files, and `fsync()` to flush the data files. `InnoDB` does not use `O_DSYNC` directly because there have been problems with it on many varieties of Unix.
- `littlesync` or `2`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `nosync` or `3`: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- `O_DIRECT` or `4`: `InnoDB` uses `O_DIRECT` (or `directio()` on Solaris) to open the data files, and uses `fsync()` to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.
- `O_DIRECT_NO_FSYNC`: `InnoDB` uses `O_DIRECT` during flushing I/O, but skips the `fsync()` system call after each write operation.

  MySQL calls `fsync()` after creating a new file, after increasing file size, and after closing a file, to ensure that file system metadata changes are synchronized. The `fsync()` system call is still skipped after each write operation.

  Data loss is possible if redo log files and data files reside on different storage devices, and an unexpected exit occurs before data file writes are flushed from a device cache that is not battery-backed. If you use or intend to use different storage devices for redo log files and data files, and your data files reside on a device with a cache that is not battery-backed, use `O_DIRECT` instead.

On platforms that support `fdatasync()` system calls, the `innodb_use_fdatasync` variable permits `innodb_flush_method` options that use `fsync()` to use `fdatasync()` instead. An `fdatasync()` system call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

The `innodb_flush_method` options for Windows systems include:

- `unbuffered` or `0`: `InnoDB` uses non-buffered I/O.

  > Note
  >
  > Running MySQL server on a 4K sector hard drive on Windows is not supported with `unbuffered`. The workaround is to use `innodb_flush_method=normal`.

- `normal` or `1`: `InnoDB` uses buffered I/O.

How each setting affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the `Innodb_data_fsyncs` status variable to see the overall number of `fsync()` calls (or `fdatasync()` calls if `innodb_use_fdatasync` is enabled) for each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, `O_DIRECT` can help to avoid double buffering between the `InnoDB` buffer pool and the operating system file system cache. On some systems where `InnoDB` data and log files are located on a SAN, the default value or `O_DSYNC` might be faster for a read-heavy workload with mostly `SELECT` statements. Always test this parameter with hardware and workload that reflect your production environment. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_flush_neighbors`

| | |
|---|---|
| **Command-Line Format** | `--innodb-flush-neighbors=#` |
| **System Variable** | `innodb_flush_neighbors` |
| **Scope** | Global |
| **Dynamic** | Yes |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | 0<br>0 |
| **Valid Values** | 1<br>2 |

Specifies whether flushing a page from the InnoDB buffer pool also flushes other dirty pages in the same extent.

A setting of 0 disables innodb_flush_neighbors. Dirty pages in the same extent are not flushed.

A setting of 1 flushes contiguous dirty pages in the same extent.

A setting of 2 flushes dirty pages in the same extent.

When the table data is stored on a traditional HDD storage device, flushing such neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can set this option to 0 to spread out write operations. For related information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”.

innodb_flush_sync

| | |
|---|---|
| **Command-Line Format** | --innodb-flush-sync[={OFF\|ON}] |
| **System Variable** | innodb_flush_sync |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

The innodb_flush_sync variable, which is enabled by default, causes the innodb_io_capacity and innodb_io_capacity_max settings to be ignored during bursts of I/O activity that occur at checkpoints. To adhere to the I/O rate defined by innodb_io_capacity and innodb_io_capacity_max, disable innodb_flush_sync.

For information about configuring the innodb_flush_sync variable, see Section 17.8.7, “Configuring InnoDB I/O Capacity”.

innodb_flushing_avg_loops

| | |
|---|---|
| **Command-Line Format** | --innodb-flushing-avg-loops=# |
| **System Variable** | innodb_flushing_avg_loops |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 30 |
| **Minimum Value** | 1 |
| **Maximum Value** | 1000 |

Number of iterations for which InnoDB keeps the previously calculated snapshot of the flushing state, controlling how quickly adaptive flushing responds to changing workloads. Increasing the value makes the rate of flush operations change smoothly and gradually as the workload changes. Decreasing the value makes adaptive flushing adjust quickly to workload changes, which can cause spikes in flushing activity if the workload increases and decreases suddenly.

For related information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”.

innodb_force_load_corrupted

| | |
|---|---|
| **Command-Line Format** | --innodb-force-load-corrupted[={OFF\|ON}] |
| **System Variable** | innodb_force_load_corrupted |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Permits InnoDB to load tables at startup that are marked as corrupted. Use only during troubleshooting, to recover data that is otherwise inaccessible. When troubleshooting is complete, disable this setting and restart the server.

innodb_force_recovery

| | |
|---|---|
| **Command-Line Format** | --innodb-force-recovery=# |
| **System Variable** | innodb_force_recovery |
| **Scope** | Global |

| | |
|---|---|
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 6 |

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. For the meanings of these values and important information about `innodb_force_recovery`, see Section 17.20.3, “Forcing InnoDB Recovery”.

> Warning
>
> Only set this variable to a value greater than 0 in an emergency situation so that you can start `InnoDB` and dump your tables. As a safety measure, `InnoDB` prevents `INSERT`, `UPDATE`, or `DELETE` operations when `innodb_force_recovery` is greater than 0. An `innodb_force_recovery` setting of 4 or greater places `InnoDB` into read-only mode.
>
> These restrictions may cause replication administration commands to fail with an error, as replication stores the replica status logs in `InnoDB` tables.

`innodb_fsync_threshold`

| | |
|---|---|
| **Command-Line Format** | `--innodb-fsync-threshold=#` |
| **System Variable** | `innodb_fsync_threshold` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 2**64-1 |

By default, when `InnoDB` creates a new data file, such as a new log file or tablespace file, the file is fully written to the operating system cache before it is flushed to disk, which can cause a large amount of disk write activity to occur at once. To force smaller, periodic flushes of data from the operating system cache, you can use the `innodb_fsync_threshold` variable to define a threshold value, in bytes. When the byte threshold is reached, the contents of the operating system cache are flushed to disk. The default value of 0 forces the default behavior, which is to flush data to disk only after a file is fully written to the cache.

Specifying a threshold to force smaller, periodic flushes may be beneficial in cases where multiple MySQL instances use the same storage devices. For example, creating a new MySQL instance and its associated data files could cause large surges of disk write activity, impeding the performance of other MySQL instances that use the same storage devices. Configuring a threshold helps avoid such surges in write activity.

`innodb_ft_aux_table`

| | |
|---|---|
| **System Variable** | `innodb_ft_aux_table` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |

Specifies the qualified name of an `InnoDB` table containing a `FULLTEXT` index. This variable is intended for diagnostic purposes and can only be set at runtime. For example:

```
SET GLOBAL innodb_ft_aux_table = 'test/t1';
```

After you set this variable to a name in the format *`db_name/table_name`*, the `INFORMATION_SCHEMA` tables `INNODB_FT_INDEX_TABLE`, `INNODB_FT_INDEX_CACHE`, `INNODB_FT_CONFIG`, `INNODB_FT_DELETED`, and `INNODB_FT_BEING_DELETED` show information about the search index for the specified table.

For more information, see Section 17.15.4, “InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables”.

`innodb_ft_cache_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-ft-cache-size=#` |
| **System Variable** | `innodb_ft_cache_size` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 8000000 |
| **Minimum Value** | 1600000 |
| **Maximum Value** | 80000000 |
| **Unit** | bytes |

The memory allocated, in bytes, for the InnoDB FULLTEXT search index cache, which holds a parsed document in memory while creating an InnoDB FULLTEXT index. Index inserts and updates are only committed to disk when the innodb_ft_cache_size size limit is reached. innodb_ft_cache_size defines the cache size on a per table basis. To set a global limit for all tables, see innodb_ft_total_cache_size.

For more information, see InnoDB Full-Text Index Cache.

### innodb_ft_enable_diag_print

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-enable-diag-print[={OFF\|ON}] |
| **System Variable** | innodb_ft_enable_diag_print |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Whether to enable additional full-text search (FTS) diagnostic output. This option is primarily intended for advanced FTS debugging and is not of interest to most users. Output is printed to the error log and includes information such as:

FTS index sync progress (when the FTS cache limit is reached). For example:

```
FTS SYNC for table test, deleted count: 100 size: 10000 bytes
SYNC words: 100
```

FTS optimize progress. For example:

```
FTS start optimize test
FTS_OPTIMIZE: optimize "mysql"
FTS_OPTIMIZE: processed "mysql"
```

FTS index build progress. For example:

```
Number of doc processed: 1000
```

For FTS queries, the query parsing tree, word weight, query processing time, and memory usage are printed. For example:

```
FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000
Full Search Memory: 245666 (bytes),  Row: 10000
```

### innodb_ft_enable_stopword

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-enable-stopword[={OFF\|ON}] |
| **System Variable** | innodb_ft_enable_stopword |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Specifies that a set of stopwords is associated with an InnoDB FULLTEXT index at the time the index is created. If the innodb_ft_user_stopword_table option is set, the stopwords are taken from that table. Else, if the innodb_ft_server_stopword_table option is set, the stopwords are taken from that table. Otherwise, a built-in set of default stopwords is used.

For more information, see Section 14.9.4, “Full-Text Stopwords”.

### innodb_ft_max_token_size

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-max-token-size=# |
| **System Variable** | innodb_ft_max_token_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 84 |
| **Minimum Value** | 10 |
| **Maximum Value** | 84 |

Maximum character length of words that are stored in an InnoDB FULLTEXT index. Setting a limit on this value reduces the size of the index, thus speeding up queries, by omitting long keywords or arbitrary collections of letters that are not real words and are not likely to be search terms.

For more information, see Section 14.9.6, “Fine-Tuning MySQL Full-Text Search”.

### innodb_ft_min_token_size

| | |
|---|---|
| **Command-Line Format** | `--innodb-ft-min-token-size=#` |
| **System Variable** | `innodb_ft_min_token_size` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `3` |
| **Minimum Value** | `0` |
| **Maximum Value** | `16` |

Minimum length of words that are stored in an `InnoDB` `FULLTEXT` index. Increasing this value reduces the size of the index, thus speeding up queries, by omitting common words that are unlikely to be significant in a search context, such as the English words “a” and “to”. For content using a CJK (Chinese, Japanese, Korean) character set, specify a value of 1.

For more information, see Section 14.9.6, “Fine-Tuning MySQL Full-Text Search”.

### innodb_ft_num_word_optimize

| | |
|---|---|
| **Command-Line Format** | `--innodb-ft-num-word-optimize=#` |
| **System Variable** | `innodb_ft_num_word_optimize` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `2000` |
| **Minimum Value** | `1000` |
| **Maximum Value** | `10000` |

Number of words to process during each `OPTIMIZE TABLE` operation on an `InnoDB` `FULLTEXT` index. Because a bulk insert or update operation to a table containing a full-text search index could require substantial index maintenance to incorporate all changes, you might do a series of `OPTIMIZE TABLE` statements, each picking up where the last left off.

For more information, see Section 14.9.6, “Fine-Tuning MySQL Full-Text Search”.

### innodb_ft_result_cache_limit

| | |
|---|---|
| **Command-Line Format** | `--innodb-ft-result-cache-limit=#` |
| **System Variable** | `innodb_ft_result_cache_limit` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `2000000000` |
| **Minimum Value** | `1000000` |
| **Maximum Value** | `2**32-1` |
| **Unit** | bytes |

The `InnoDB` full-text search query result cache limit (defined in bytes) per full-text search query or per thread. Intermediate and final `InnoDB` full-text search query results are handled in memory. Use `innodb_ft_result_cache_limit` to place a size limit on the full-text search query result cache to avoid excessive memory consumption in case of very large `InnoDB` full-text search query results (millions or hundreds of millions of rows, for example). Memory is allocated as required when a full-text search query is processed. If the result cache size limit is reached, an error is returned indicating that the query exceeds the maximum allowed memory.

The maximum value of `innodb_ft_result_cache_limit` for all platform types and bit sizes is 2**32-1.

### innodb_ft_server_stopword_table

| | |
|---|---|
| **Command-Line Format** | `--innodb-ft-server-stopword-table=db_name/table_name` |
| **System Variable** | `innodb_ft_server_stopword_table` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |
| **Default Value** | `NULL` |

This option is used to specify your own `InnoDB` `FULLTEXT` index stopword list for all `InnoDB` tables. To configure your own stopword list for a specific `InnoDB` table, use `innodb_ft_user_stopword_table`.

Set `innodb_ft_server_stopword_table` to the name of the table containing a list of stopwords, in the format *`db_name/table_name`*.

The stopword table must exist before you configure innodb_ft_server_stopword_table. innodb_ft_enable_stopword must be enabled and innodb_ft_server_stopword_table option must be configured before you create the FULLTEXT index.

The stopword table must be an InnoDB table, containing a single VARCHAR column named value.

For more information, see Section 14.9.4, “Full-Text Stopwords”.

### innodb_ft_sort_pll_degree

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-sort-pll-degree=# |
| **System Variable** | innodb_ft_sort_pll_degree |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 2 |
| **Minimum Value** | 1 |
| **Maximum Value** | 16 |

Number of threads used in parallel to index and tokenize text in an InnoDB FULLTEXT index when building a search index.

For related information, see Section 17.6.2.4, “InnoDB Full-Text Indexes”, and innodb_sort_buffer_size.

### innodb_ft_total_cache_size

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-total-cache-size=# |
| **System Variable** | innodb_ft_total_cache_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 640000000 |
| **Minimum Value** | 32000000 |
| **Maximum Value** | 1600000000 |
| **Unit** | bytes |

The total memory allocated, in bytes, for the InnoDB full-text search index cache for all tables. Creating numerous tables, each with a FULLTEXT search index, could consume a significant portion of available memory. innodb_ft_total_cache_size defines a global memory limit for all full-text search indexes to help avoid excessive memory consumption. If the global limit is reached by an index operation, a forced sync is triggered.

For more information, see InnoDB Full-Text Index Cache.

### innodb_ft_user_stopword_table

| | |
|---|---|
| **Command-Line Format** | --innodb-ft-user-stopword-table=db_name/table_name |
| **System Variable** | innodb_ft_user_stopword_table |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |
| **Default Value** | NULL |

This option is used to specify your own InnoDB FULLTEXT index stopword list on a specific table. To configure your own stopword list for all InnoDB tables, use innodb_ft_server_stopword_table.

Set innodb_ft_user_stopword_table to the name of the table containing a list of stopwords, in the format *db_name/table_name*.

The stopword table must exist before you configure innodb_ft_user_stopword_table. innodb_ft_enable_stopword must be enabled and innodb_ft_user_stopword_table must be configured before you create the FULLTEXT index.

The stopword table must be an InnoDB table, containing a single VARCHAR column named value.

For more information, see Section 14.9.4, “Full-Text Stopwords”.

### innodb_idle_flush_pct

| | |
|---|---|
| **Command-Line Format** | --innodb-idle-flush-pct=# |
| **System Variable** | innodb_idle_flush_pct |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 100 |

| | |
|---|---|
| **Minimum Value** | `0` |
| **Maximum Value** | `100` |

Limits page flushing when `InnoDB` is idle. The `innodb_idle_flush_pct` value is a percentage of the `innodb_io_capacity` setting, which defines the number of I/O operations per second available to `InnoDB`. For more information, see Limiting Buffer Flushing During Idle Periods.

`innodb_io_capacity`

| | |
|---|---|
| **Command-Line Format** | `--innodb-io-capacity=#` |
| **System Variable** | `innodb_io_capacity` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `10000` |
| **Minimum Value** | `100` |
| **Maximum Value (64-bit platforms, 8.4.0)** | `2**64-1` |
| **Maximum Value** | `2**32-1` |

The `innodb_io_capacity` variable defines the number of I/O operations per second (IOPS) available to `InnoDB` background tasks, such as flushing pages from the buffer pool and merging data from the change buffer.

For information about configuring the `innodb_io_capacity` variable, see Section 17.8.7, “Configuring InnoDB I/O Capacity”.

`innodb_io_capacity_max`

| | |
|---|---|
| **Command-Line Format** | `--innodb-io-capacity-max=#` |
| **System Variable** | `innodb_io_capacity_max` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `2 * innodb_io_capacity` |
| **Minimum Value** | `100` |
| **Maximum Value** | `2**32-1` |

If flushing activity falls behind, `InnoDB` can flush more aggressively, at a higher rate of I/O operations per second (IOPS) than defined by the `innodb_io_capacity` variable. The `innodb_io_capacity_max` variable defines a maximum number of IOPS performed by `InnoDB` background tasks in such situations. This option does not control `innodb_flush_sync` behavior.

The default value is twice the value of `innodb_io_capacity`.

For information about configuring the `innodb_io_capacity_max` variable, see Section 17.8.7, “Configuring InnoDB I/O Capacity”.

`innodb_limit_optimistic_insert_debug`

| | |
|---|---|
| **Command-Line Format** | `--innodb-limit-optimistic-insert-debug=#` |
| **System Variable** | `innodb_limit_optimistic_insert_debug` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `0` |
| **Minimum Value** | `0` |
| **Maximum Value** | `2**32-1` |

Limits the number of records per B-tree page. A default value of 0 means that no limit is imposed. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

`innodb_lock_wait_timeout`

| | |
|---|---|
| **Command-Line Format** | `--innodb-lock-wait-timeout=#` |
| **System Variable** | `innodb_lock_wait_timeout` |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `50` |
| **Minimum Value** | `1` |
| **Maximum Value** | `1073741824` |
| **Unit** | seconds |

The length of time in seconds an InnoDB transaction waits for a row lock before giving up. The default value is 50 seconds. A transaction that tries to access a row that is locked by another InnoDB transaction waits at most this many seconds for write access to the row before issuing the following error:

```
ERROR 1205 (HY000): Lock wait timeout exceeded; try restarting transaction
```

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the `--innodb-rollback-on-timeout` option. See also Section 17.20.5, “InnoDB Error Handling”.

You might decrease this value for highly interactive applications or OLTP systems, to display user feedback quickly or put the update into a queue for processing later. You might increase this value for long-running back-end operations, such as a transform step in a data warehouse that waits for other large insert or update operations to finish.

`innodb_lock_wait_timeout` applies to InnoDB row locks. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks when `innodb_deadlock_detect` is enabled (the default) because InnoDB detects deadlocks immediately and rolls back one of the deadlocked transactions. When `innodb_deadlock_detect` is disabled, InnoDB relies on `innodb_lock_wait_timeout` for transaction rollback when a deadlock occurs. See Section 17.7.5.2, “Deadlock Detection”.

`innodb_lock_wait_timeout` can be set at runtime with the `SET GLOBAL` or `SET SESSION` statement. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 7.1.9.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_lock_wait_timeout`, which affects only that client.

`innodb_log_buffer_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-buffer-size=#` |
| **System Variable** | `innodb_log_buffer_size` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `67108864` |
| **Minimum Value** | `1048576` |
| **Maximum Value** | `4294967295` |

The size in bytes of the buffer that InnoDB uses to write to the log files on disk. The default is 64MB. A large log buffer enables large transactions to run without the need to write the log to disk before the transactions commit. Thus, if you have transactions that update, insert, or delete many rows, making the log buffer larger saves disk I/O. For related information, see Memory Configuration, and Section 10.5.4, “Optimizing InnoDB Redo Logging”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_log_checkpoint_fuzzy_now`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-checkpoint-fuzzy-now[={OFF\|ON}]` |
| **System Variable** | `innodb_log_checkpoint_fuzzy_now` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enable this debug option to force InnoDB to write a fuzzy checkpoint. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

`innodb_log_checkpoint_now`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-checkpoint-now[={OFF\|ON}]` |
| **System Variable** | `innodb_log_checkpoint_now` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enable this debug option to force InnoDB to write a checkpoint. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

`innodb_log_checksums`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-checksums[={OFF\|ON}]` |
| **System Variable** | `innodb_log_checksums` |
| **Scope** | Global |
| **Dynamic** | Yes |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Enables or disables checksums for redo log pages.

innodb_log_checksums=ON enables the CRC-32C checksum algorithm for redo log pages. When innodb_log_checksums is disabled, the contents of the redo log page checksum field are ignored.

Checksums on the redo log header page and redo log checkpoint pages are never disabled.

innodb_log_compressed_pages

| | |
|---|---|
| **Command-Line Format** | --innodb-log-compressed-pages[={OFF\|ON}] |
| **System Variable** | innodb_log_compressed_pages |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Specifies whether images of re-compressed pages are written to the redo log. Re-compression may occur when changes are made to compressed data.

innodb_log_compressed_pages is enabled by default to prevent corruption that could occur if a different version of the zlib compression algorithm is used during recovery. If you are certain that the zlib version is not subject to change, you can disable innodb_log_compressed_pages to reduce redo log generation for workloads that modify compressed data.

To measure the effect of enabling or disabling innodb_log_compressed_pages, compare redo log generation for both settings under the same workload. Options for measuring redo log generation include observing the Log sequence number (LSN) in the LOG section of SHOW ENGINE INNODB STATUS output, or monitoring Innodb_os_log_written status for the number of bytes written to the redo log files.

For related information, see Section 17.9.1.6, “Compression for OLTP Workloads”.

innodb_log_file_size

| | |
|---|---|
| **Command-Line Format** | --innodb-log-file-size=# |
| **Deprecated** | Yes |
| **System Variable** | innodb_log_file_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 50331648 |
| **Minimum Value** | 4194304 |
| **Maximum Value** | 512GB / innodb_log_files_in_group |
| **Unit** | bytes |

Note

innodb_log_file_size and innodb_log_files_in_group have been superseded by innodb_redo_log_capacity; see Section 17.6.5, “Redo Log”.

The size in bytes of each log file in a log group. The combined size of log files (innodb_log_file_size * innodb_log_files_in_group) cannot exceed a maximum value that is slightly less than 512GB. A pair of 255 GB log files, for example, approaches the limit but does not exceed it. The default value is 48MB.

Generally, the combined size of the log files should be large enough that the server can smooth out peaks and troughs in workload activity, which often means that there is enough redo log space to handle more than an hour of write activity. The larger the value, the less checkpoint flush activity is required in the buffer pool, saving disk I/O. Larger log files also make crash recovery slower.

The minimum innodb_log_file_size is 4MB.

For related information, see Redo Log Configuration. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

If innodb_dedicated_server is enabled, the innodb_log_file_size value is automatically configured if it is not explicitly defined. For more information, see Section 17.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

innodb_log_files_in_group

| | |
|---|---|
| **Command-Line Format** | --innodb-log-files-in-group=# |
| **Deprecated** | Yes |
| **System Variable** | innodb_log_files_in_group |
| **Scope** | Global |
| **Dynamic** | No |

| | |
|---|---|
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 2 |
| **Minimum Value** | 2 |
| **Maximum Value** | 100 |

Note

`innodb_log_file_size` and `innodb_log_files_in_group` have been superseded by `innodb_redo_log_capacity`; see Section 17.6.5, “Redo Log”.

The number of log files in the log group. `InnoDB` writes to the files in a circular fashion. The default (and recommended) value is 2. The location of the files is specified by `innodb_log_group_home_dir`. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) can be up to 512GB.

For related information, see Redo Log Configuration.

If `innodb_dedicated_server` is enabled, `innodb_log_files_in_group` is automatically configured if it is not explicitly defined. For more information, see Section 17.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

`innodb_log_group_home_dir`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-group-home-dir=dir_name` |
| **System Variable** | `innodb_log_group_home_dir` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |

The directory path to the `InnoDB` redo log files.

For related information, see Redo Log Configuration.

`innodb_log_spin_cpu_abs_lwm`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-spin-cpu-abs-lwm=#` |
| **System Variable** | `innodb_log_spin_cpu_abs_lwm` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 80 |
| **Minimum Value** | 0 |
| **Maximum Value** | 4294967295 |

Defines the minimum amount of CPU usage below which user threads no longer spin while waiting for flushed redo. The value is expressed as a sum of CPU core usage. For example, The default value of 80 is 80% of a single CPU core. On a system with a multi-core processor, a value of 150 represents 100% usage of one CPU core plus 50% usage of a second CPU core.

For related information, see Section 10.5.4, “Optimizing InnoDB Redo Logging”.

`innodb_log_spin_cpu_pct_hwm`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-spin-cpu-pct-hwm=#` |
| **System Variable** | `innodb_log_spin_cpu_pct_hwm` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 50 |
| **Minimum Value** | 0 |
| **Maximum Value** | 100 |

Defines the maximum amount of CPU usage above which user threads no longer spin while waiting for flushed redo. The value is expressed as a percentage of the combined total processing power of all CPU cores. The default value is 50%. For example, 100% usage of two CPU cores is 50% of the combined CPU processing power on a server with four CPU cores.

The `innodb_log_spin_cpu_pct_hwm` variable respects processor affinity. For example, if a server has 48 cores but the **mysqld** process is pinned to only four CPU cores, the other 44 CPU cores are ignored.

For related information, see Section 10.5.4, “Optimizing InnoDB Redo Logging”.

`innodb_log_wait_for_flush_spin_hwm`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-wait-for-flush-spin-hwm=#` |
| **System Variable** | `innodb_log_wait_for_flush_spin_hwm` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `400` |
| **Minimum Value** | `0` |
| **Maximum Value (64-bit platforms, 8.4.0)** | `2**64-1` |
| **Maximum Value** | `2**32-1` |
| **Unit** | microseconds |

Defines the maximum average log flush time beyond which user threads no longer spin while waiting for flushed redo. The default value is 400 microseconds.

For related information, see Section 10.5.4, “Optimizing InnoDB Redo Logging”.

`innodb_log_write_ahead_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-write-ahead-size=#` |
| **System Variable** | `innodb_log_write_ahead_size` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `8192` |
| **Minimum Value** | `512 (log file block size)` |
| **Maximum Value** | `Equal to innodb_page_size` |
| **Unit** | bytes |

Defines the write-ahead block size for the redo log, in bytes. To avoid “read-on-write”, set `innodb_log_write_ahead_size` to match the operating system or file system cache block size. The default setting is 8192 bytes. Read-on-write occurs when redo log blocks are not entirely cached to the operating system or file system due to a mismatch between write-ahead block size for the redo log and operating system or file system cache block size.

Valid values for `innodb_log_write_ahead_size` are multiples of the `InnoDB` log file block size ($2^n$). The minimum value is the `InnoDB` log file block size (512). Write-ahead does not occur when the minimum value is specified. The maximum value is equal to the `innodb_page_size` value. If you specify a value for `innodb_log_write_ahead_size` that is larger than the `innodb_page_size` value, the `innodb_log_write_ahead_size` setting is truncated to the `innodb_page_size` value.

Setting the `innodb_log_write_ahead_size` value too low in relation to the operating system or file system cache block size results in “read-on-write”. Setting the value too high may have a slight impact on `fsync` performance for log file writes due to several blocks being written at once.

For related information, see Section 10.5.4, “Optimizing InnoDB Redo Logging”.

`innodb_log_writer_threads`

| | |
|---|---|
| **Command-Line Format** | `--innodb-log-writer-threads[={OFF\|ON}]` |
| **System Variable** | `innodb_log_writer_threads` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

Enables dedicated log writer threads for writing redo log records from the log buffer to the system buffers and flushing the system buffers to the redo log files. Dedicated log writer threads can improve performance on high-concurrency systems, but for low-concurrency systems, disabling dedicated log writer threads provides better performance.

For more information, see Section 10.5.4, “Optimizing InnoDB Redo Logging”.

`innodb_lru_scan_depth`

| | |
|---|---|
| **Command-Line Format** | `--innodb-lru-scan-depth=#` |
| **System Variable** | `innodb_lru_scan_depth` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `1024` |
| **Minimum Value** | `100` |
| **Maximum Value (64-bit platforms, 8.4.0)** | `2**64-1` |
| **Maximum Value** | `2**32-1` |

A parameter that influences the algorithms and heuristics for the flush operation for the InnoDB buffer pool. Primarily of interest to performance experts tuning I/O-intensive workloads. It specifies, per buffer pool instance, how far down the buffer pool LRU page list the page cleaner thread scans looking for dirty pages to flush. This is a background operation performed once per second.

A setting smaller than the default is generally suitable for most workloads. A value that is much higher than necessary may impact performance. Only consider increasing the value if you have spare I/O capacity under a typical workload. Conversely, if a write-intensive workload saturates your I/O capacity, decrease the value, especially in the case of a large buffer pool.

When tuning `innodb_lru_scan_depth`, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting `innodb_lru_scan_depth` when changing the number of buffer pool instances, since `innodb_lru_scan_depth` * `innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

For related information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_max_dirty_pages_pct`

| | |
|---|---|
| **Command-Line Format** | `--innodb-max-dirty-pages-pct=#` |
| **System Variable** | `innodb_max_dirty_pages_pct` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Numeric |
| **Default Value** | `90` |
| **Minimum Value** | `0` |
| **Maximum Value** | `99.999` |

InnoDB tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value.

The `innodb_max_dirty_pages_pct` setting establishes a target for flushing activity. It does not affect the rate of flushing. For information about managing the rate of flushing, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”.

For related information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_max_dirty_pages_pct_lwm`

| | |
|---|---|
| **Command-Line Format** | `--innodb-max-dirty-pages-pct-lwm=#` |
| **System Variable** | `innodb_max_dirty_pages_pct_lwm` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Numeric |
| **Default Value** | `10` |
| **Minimum Value** | `0` |
| **Maximum Value** | `99.999` |

Defines a low water mark representing the percentage of dirty pages at which preflushing is enabled to control the dirty page ratio. A value of 0 disables the pre-flushing behavior entirely. The configured value should always be lower than the `innodb_max_dirty_pages_pct` value. For more information, see Section 17.8.3.5, “Configuring Buffer Pool Flushing”.

`innodb_max_purge_lag`

| | |
|---|---|
| **Command-Line Format** | `--innodb-max-purge-lag=#` |
| **System Variable** | `innodb_max_purge_lag` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `0` |
| **Minimum Value** | `0` |
| **Maximum Value** | `4294967295` |

Defines the desired maximum purge lag. If this value is exceeded, a delay is imposed on `INSERT`, `UPDATE`, and `DELETE` operations to allow time for purge to catch up. The default value is 0, which means there is no maximum purge lag and no delay.

For more information, see Section 17.8.9, “Purge Configuration”.

`innodb_max_purge_lag_delay`

| | |
|---|---|
| **Command-Line Format** | `--innodb-max-purge-lag-delay=#` |
| **System Variable** | `innodb_max_purge_lag_delay` |

| | |
|---|---|
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `0` |
| **Minimum Value** | `0` |
| **Maximum Value** | `10000000` |
| **Unit** | microseconds |

Specifies the maximum delay in microseconds for the delay imposed when the `innodb_max_purge_lag` threshold is exceeded. The specified `innodb_max_purge_lag_delay` value is an upper limit on the delay period calculated by the `innodb_max_purge_lag` formula.

For more information, see Section 17.8.9, “Purge Configuration”.

`innodb_max_undo_log_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-max-undo-log-size=#` |
| **System Variable** | `innodb_max_undo_log_size` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `1073741824` |
| **Minimum Value** | `10485760` |
| **Maximum Value** | `2**64-1` |
| **Unit** | bytes |

Defines a threshold size for undo tablespaces. If an undo tablespace exceeds the threshold, it can be marked for truncation when `innodb_undo_log_truncate` is enabled. The default value is 1073741824 bytes (1024 MiB).

For more information, see Truncating Undo Tablespaces.

`innodb_merge_threshold_set_all_debug`

| | |
|---|---|
| **Command-Line Format** | `--innodb-merge-threshold-set-all-debug=#` |
| **System Variable** | `innodb_merge_threshold_set_all_debug` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `50` |
| **Minimum Value** | `1` |
| **Maximum Value** | `50` |

Defines a page-full percentage value for index pages that overrides the current `MERGE_THRESHOLD` setting for all indexes that are currently in the dictionary cache. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option. For related information, see Section 17.8.11, “Configuring the Merge Threshold for Index Pages”.

`innodb_monitor_disable`

| | |
|---|---|
| **Command-Line Format** | `--innodb-monitor-disable={counter\|module\|pattern\|all}` |
| **System Variable** | `innodb_monitor_disable` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |

This variable acts as a switch, disabling `InnoDB` metrics counters. Counter data may be queried using the Information Schema `INNODB_METRICS` table. For usage information, see Section 17.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

`innodb_monitor_disable='latch'` disables statistics collection for `SHOW ENGINE INNODB MUTEX`. For more information, see Section 15.7.7.16, “SHOW ENGINE Statement”.

`innodb_monitor_enable`

| | |
|---|---|
| **Command-Line Format** | `--innodb-monitor-enable={counter\|module\|pattern\|all}` |
| **System Variable** | `innodb_monitor_enable` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |

This variable acts as a switch, enabling InnoDB metrics counters. Counter data may be queried using the Information Schema INNODB_METRICS table. For usage information, see Section 17.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_enable='latch' enables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 15.7.7.16, “SHOW ENGINE Statement”.

innodb_monitor_reset

| | |
|---|---|
| **Command-Line Format** | --innodb-monitor-reset={counter\|module\|pattern\|all} |
| **System Variable** | innodb_monitor_reset |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | NULL |
| **Valid Values** | counter<br>module<br>pattern<br>all |

This variable acts as a switch, resetting the count value for InnoDB metrics counters to zero. Counter data may be queried using the Information Schema INNODB_METRICS table. For usage information, see Section 17.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_reset='latch' resets statistics reported by SHOW ENGINE INNODB MUTEX. For more information, see Section 15.7.7.16, “SHOW ENGINE Statement”.

innodb_monitor_reset_all

| | |
|---|---|
| **Command-Line Format** | --innodb-monitor-reset-all={counter\|module\|pattern\|all} |
| **System Variable** | innodb_monitor_reset_all |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | NULL |
| **Valid Values** | counter<br>module<br>pattern<br>all |

This variable acts as a switch, resetting all values (minimum, maximum, and so on) for InnoDB metrics counters. Counter data may be queried using the Information Schema INNODB_METRICS table. For usage information, see Section 17.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_numa_interleave

| | |
|---|---|
| **Command-Line Format** | --innodb-numa-interleave[={OFF\|ON}] |
| **System Variable** | innodb_numa_interleave |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Enables the NUMA interleave memory policy for allocation of the InnoDB buffer pool. When innodb_numa_interleave is enabled, the NUMA memory policy is set to MPOL_INTERLEAVE for the **mysqld** process. After the InnoDB buffer pool is allocated, the NUMA memory policy is set back to MPOL_DEFAULT. For the innodb_numa_interleave option to be available, MySQL must be compiled on a NUMA-enabled Linux system. The default value is ON if the system supports it, otherwise it defaults to OFF.

**CMake** sets the default WITH_NUMA value based on whether the current platform has NUMA support. For more information, see Section 2.8.7, “MySQL Source-Configuration Options”.

innodb_old_blocks_pct

| | |
|---|---|
| **Command-Line Format** | --innodb-old-blocks-pct=# |
| **System Variable** | innodb_old_blocks_pct |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |

| | |
|---|---|
| **Type** | Integer |
| **Default Value** | 37 |
| **Minimum Value** | 5 |
| **Maximum Value** | 95 |

Specifies the approximate percentage of the `InnoDB` buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). Often used in combination with `innodb_old_blocks_time`.

For more information, see Section 17.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 17.5.1, “Buffer Pool”.

### `innodb_old_blocks_time`

| | |
|---|---|
| **Command-Line Format** | `--innodb-old-blocks-time=#` |
| **System Variable** | `innodb_old_blocks_time` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `1000` |
| **Minimum Value** | `0` |
| **Maximum Value** | `2**32-1` |
| **Unit** | milliseconds |

Non-zero values protect against the buffer pool being filled by data that is referenced only for a brief period, such as during a full table scan. Increasing this value offers more protection against full table scans interfering with data cached in the buffer pool.

Specifies how long in milliseconds a block inserted into the old sublist must stay there after its first access before it can be moved to the new sublist. If the value is 0, a block inserted into the old sublist moves immediately to the new sublist the first time it is accessed, no matter how soon after insertion the access occurs. If the value is greater than 0, blocks remain in the old sublist until an access occurs at least that many milliseconds after the first access. For example, a value of 1000 causes blocks to stay in the old sublist for 1 second after the first access before they become eligible to move to the new sublist.

The default value is 1000.

This variable is often used in combination with `innodb_old_blocks_pct`. For more information, see Section 17.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 17.5.1, “Buffer Pool”.

### `innodb_online_alter_log_max_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-online-alter-log-max-size=#` |
| **System Variable** | `innodb_online_alter_log_max_size` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `134217728` |
| **Minimum Value** | `65536` |
| **Maximum Value** | `2**64-1` |
| **Unit** | bytes |

Specifies an upper limit in bytes on the size of the temporary log files used during online DDL operations for `InnoDB` tables. There is one such log file for each index being created or table being altered. This log file stores data inserted, updated, or deleted in the table during the DDL operation. The temporary log file is extended when needed by the value of `innodb_sort_buffer_size`, up to the maximum specified by `innodb_online_alter_log_max_size`. If a temporary log file exceeds the upper size limit, the `ALTER TABLE` operation fails and all uncommitted concurrent DML operations are rolled back. Thus, a large value for this option allows more DML to happen during an online DDL operation, but also extends the period of time at the end of the DDL operation when the table is locked to apply the data from the log.

### `innodb_open_files`

| | |
|---|---|
| **Command-Line Format** | `--innodb-open-files=#` |
| **System Variable** | `innodb_open_files` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `-1` (signifies autosizing; do not assign this literal value) |
| **Minimum Value** | `10` |
| **Maximum Value** | `2147483647` |

Specifies the maximum number of files that `InnoDB` can have open at one time. The minimum value is 10. If `innodb_file_per_table` is disabled, the default value is 300; otherwise, the default value is 300 or the `table_open_cache` setting, whichever is higher.

The innodb_open_files limit can be set at runtime using a SELECT innodb_set_open_files_limit(N) statement, where N is the desired innodb_open_files limit; for example:

```
mysql> SELECT innodb_set_open_files_limit(1000);
```

The statement executes a stored procedure that sets the new limit. If the procedure is successful, it returns the value of the newly set limit; otherwise, a failure message is returned.

It is not permitted to set innodb_open_files using a SET statement. To set innodb_open_files at runtime, use the SELECT innodb_set_open_files_limit(N) statement described above.

Setting innodb_open_files=default is not supported. Only integer values are permitted.

To prevent non-LRU managed files from consuming the entire innodb_open_files limit, non-LRU managed files are limited to 90 percent of this limit, which reserves 10 percent of it for LRU managed files.

innodb_optimize_fulltext_only

| | |
|---|---|
| **Command-Line Format** | --innodb-optimize-fulltext-only[={OFF\|ON}] |
| **System Variable** | innodb_optimize_fulltext_only |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Changes the way OPTIMIZE TABLE operates on InnoDB tables. Intended to be enabled temporarily, during maintenance operations for InnoDB tables with FULLTEXT indexes.

By default, OPTIMIZE TABLE reorganizes data in the clustered index of the table. When this option is enabled, OPTIMIZE TABLE skips the reorganization of table data, and instead processes newly added, deleted, and updated token data for InnoDB FULLTEXT indexes. For more information, see Optimizing InnoDB Full-Text Indexes.

innodb_page_cleaners

| | |
|---|---|
| **Command-Line Format** | --innodb-page-cleaners=# |
| **System Variable** | innodb_page_cleaners |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | innodb_buffer_pool_instances |
| **Minimum Value** | 1 |
| **Maximum Value** | 64 |

The number of page cleaner threads that flush dirty pages from buffer pool instances. Page cleaner threads perform flush list and LRU flushing. When there are multiple page cleaner threads, buffer pool flushing tasks for each buffer pool instance are dispatched to idle page cleaner threads. The innodb_page_cleaners default value is set to the same value as innodb_buffer_pool_instances. If the specified number of page cleaner threads exceeds the number of buffer pool instances, then innodb_page_cleaners is automatically set to the same value as innodb_buffer_pool_instances.

If your workload is write-IO bound when flushing dirty pages from buffer pool instances to data files, and if your system hardware has available capacity, increasing the number of page cleaner threads may help improve write-IO throughput.

Multithreaded page cleaner support extends to shutdown and recovery phases.

The setpriority() system call is used on Linux platforms where it is supported, and where the **mysqld** execution user is authorized to give page_cleaner threads priority over other MySQL and InnoDB threads to help page flushing keep pace with the current workload. setpriority() support is indicated by this InnoDB startup message:

```
[Note] InnoDB: If the mysqld execution user is authorized, page cleaner
thread priority can be changed. See the man page of setpriority().
```

For systems where server startup and shutdown is not managed by systemd, **mysqld** execution user authorization can be configured in /etc/security/limits.conf. For example, if **mysqld** is run under the mysql user, you can authorize the mysql user by adding these lines to /etc/security/limits.conf:

```
mysql              hard    nice       -20
mysql              soft    nice       -20
```

For systemd managed systems, the same can be achieved by specifying LimitNICE=-20 in a localized systemd configuration file. For example, create a file named override.conf in /etc/systemd/system/mysqld.service.d/override.conf and add this entry:

```
[Service]
LimitNICE=-20
```

After creating or changing `override.conf`, reload the systemd configuration, then tell systemd to restart the MySQL service:

```
systemctl daemon-reload
systemctl restart mysqld  # RPM platforms
systemctl restart mysql   # Debian platforms
```

For more information about using a localized systemd configuration file, see Configuring systemd for MySQL.

After authorizing the **mysqld** execution user, use the **cat** command to verify the configured `Nice` limits for the **mysqld** process:

```
$> cat /proc/mysqld_pid/limits | grep nice
Max nice priority         18446744073709551596 18446744073709551596
```

`innodb_page_size`

| | |
|---|---|
| **Command-Line Format** | `--innodb-page-size=#` |
| **System Variable** | `innodb_page_size` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | `16384`<br>`4096`<br>`8192` |
| **Valid Values** | `16384`<br>`32768`<br>`65536` |

Specifies the page size for `InnoDB` tablespaces. Values can be specified in bytes or kilobytes. For example, a 16 kilobyte page size value can be specified as 16384, 16KB, or 16k.

`innodb_page_size` can only be configured prior to initializing the MySQL instance and cannot be changed afterward. If no value is specified, the instance is initialized using the default page size. See Section 17.8.1, “InnoDB Startup Configuration”.

For both 32KB and 64KB page sizes, the maximum row length is approximately 16000 bytes. `ROW_FORMAT=COMPRESSED` is not supported when `innodb_page_size` is set to 32KB or 64KB. For `innodb_page_size=32KB`, extent size is 2MB. For `innodb_page_size=64KB`, extent size is 4MB. `innodb_log_buffer_size` should be set to at least 16MB (the default is 64MB) when using 32KB or 64KB page sizes.

The default 16KB page size or larger is appropriate for a wide range of workloads, particularly for queries involving table scans and DML operations involving bulk updates. Smaller page sizes might be more efficient for OLTP workloads involving many small writes, where contention can be an issue when single pages contain many rows. Smaller pages might also be efficient with SSD storage devices, which typically use small block sizes. Keeping the `InnoDB` page size close to the storage device block size minimizes the amount of unchanged data that is rewritten to disk.

The minimum file size for the first system tablespace data file (`ibdata1`) differs depending on the `innodb_page_size` value. See the `innodb_data_file_path` option description for more information.

A MySQL instance using a particular `InnoDB` page size cannot use data files or log files from an instance that uses a different page size.

For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

`innodb_parallel_read_threads`

| | |
|---|---|
| **Command-Line Format** | `--innodb-parallel-read-threads=#` |
| **System Variable** | `innodb_parallel_read_threads` |
| **Scope** | Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `(available logical processors / 8), min of 4` |
| **Minimum Value** | `1` |
| **Maximum Value** | `256` |

Defines the number of threads that can be used for parallel clustered index reads. Parallel scanning of partitions is also supported. Parallel read threads can improve `CHECK TABLE` performance. `InnoDB` reads the clustered index twice during a `CHECK TABLE` operation. The second read can be performed in parallel. This feature does not apply to secondary index scans. The `innodb_parallel_read_threads` session variable must be set to a value greater than 1 for parallel clustered index reads to occur. The actual number of threads used to perform a parallel clustered index read is determined by the

innodb_parallel_read_threads setting or the number of index subtrees to scan, whichever is smaller. The pages read into the buffer pool during the scan are kept at the tail of the buffer pool LRU list so that they can be discarded quickly when free buffer pool pages are required.

The maximum number of parallel read threads (256) is the total number of threads for all client connections. If the thread limit is reached, connections fall back to using a single thread. The default value is calculated by the number of available logical processors on the system divided by 8, with a minimum default value of 4.

Before MySQL 8.4, the default value was always 4.

### innodb_print_all_deadlocks

| | |
|---|---|
| **Command-Line Format** | `--innodb-print-all-deadlocks[={OFF\|ON}]` |
| **System Variable** | innodb_print_all_deadlocks |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

When this option is enabled, information about all deadlocks in `InnoDB` user transactions is recorded in the `mysqld` error log. Otherwise, you see information about only the last deadlock, using the `SHOW ENGINE INNODB STATUS` statement. An occasional `InnoDB` deadlock is not necessarily an issue, because `InnoDB` detects the condition immediately and rolls back one of the transactions automatically. You might use this option to troubleshoot why deadlocks are occurring if an application does not have appropriate error-handling logic to detect the rollback and retry its operation. A large number of deadlocks might indicate the need to restructure transactions that issue DML or `SELECT ... FOR UPDATE` statements for multiple tables, so that each transaction accesses the tables in the same order, thus avoiding the deadlock condition.

For related information, see Section 17.7.5, “Deadlocks in InnoDB”.

### innodb_print_ddl_logs

| | |
|---|---|
| **Command-Line Format** | `--innodb-print-ddl-logs[={OFF\|ON}]` |
| **System Variable** | innodb_print_ddl_logs |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enabling this option causes MySQL to write DDL logs to `stderr`. For more information, see Viewing DDL Logs.

### innodb_purge_batch_size

| | |
|---|---|
| **Command-Line Format** | `--innodb-purge-batch-size=#` |
| **System Variable** | innodb_purge_batch_size |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `300` |
| **Minimum Value** | `1` |
| **Maximum Value** | `5000` |

Defines the number of undo log pages that purge parses and processes in one batch from the history list. In a multithreaded purge configuration, the coordinator purge thread divides innodb_purge_batch_size by innodb_purge_threads and assigns that number of pages to each purge thread. The innodb_purge_batch_size variable also defines the number of undo log pages that purge frees after every 128 iterations through the undo logs.

The innodb_purge_batch_size option is intended for advanced performance tuning in combination with the innodb_purge_threads setting. Most users need not change innodb_purge_batch_size from its default value.

For related information, see Section 17.8.9, “Purge Configuration”.

### innodb_purge_threads

| | |
|---|---|
| **Command-Line Format** | `--innodb-purge-threads=#` |
| **System Variable** | innodb_purge_threads |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `1 if # of available logical processors is <= 16; otherwise 4` |
| **Minimum Value** | `1` |

| | |
|---|---|
| **Maximum Value** | 32 |

The number of background threads devoted to the InnoDB purge operation. Increasing the value creates additional purge threads, which can improve efficiency on systems where DML operations are performed on multiple tables.

For related information, see Section 17.8.9, “Purge Configuration”.

### innodb_purge_rseg_truncate_frequency

| | |
|---|---|
| **Command-Line Format** | --innodb-purge-rseg-truncate-frequency=# |
| **System Variable** | innodb_purge_rseg_truncate_frequency |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 128 |
| **Minimum Value** | 1 |
| **Maximum Value** | 128 |

Defines the frequency with which the purge system frees rollback segments in terms of the number of times that purge is invoked. An undo tablespace cannot be truncated until its rollback segments are freed. Normally, the purge system frees rollback segments once every 128 times that purge is invoked. The default value is 128. Reducing this value increases the frequency with which the purge thread frees rollback segments.

innodb_purge_rseg_truncate_frequency is intended for use with innodb_undo_log_truncate. For more information, see Truncating Undo Tablespaces.

### innodb_random_read_ahead

| | |
|---|---|
| **Command-Line Format** | --innodb-random-read-ahead[={OFF\|ON}] |
| **System Variable** | innodb_random_read_ahead |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Enables the random read-ahead technique for optimizing InnoDB I/O.

For details about performance considerations for different types of read-ahead requests, see Section 17.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

### innodb_read_ahead_threshold

| | |
|---|---|
| **Command-Line Format** | --innodb-read-ahead-threshold=# |
| **System Variable** | innodb_read_ahead_threshold |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 56 |
| **Minimum Value** | 0 |
| **Maximum Value** | 64 |

Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer pool. If InnoDB reads at least innodb_read_ahead_threshold pages sequentially from an extent (64 pages), it initiates an asynchronous read for the entire following extent. The permissible range of values is 0 to 64. A value of 0 disables read-ahead. For the default of 56, InnoDB must read at least 56 pages sequentially from an extent to initiate an asynchronous read for the following extent.

Knowing how many pages are read through the read-ahead mechanism, and how many of these pages are evicted from the buffer pool without ever being accessed, can be useful when fine-tuning the innodb_read_ahead_threshold setting. SHOW ENGINE INNODB STATUS output displays counter information from the Innodb_buffer_pool_read_ahead and Innodb_buffer_pool_read_ahead_evicted global status variables, which report the number of pages brought into the buffer pool by read-ahead requests, and the number of such pages evicted from the buffer pool without ever being accessed, respectively. The status variables report global values since the last server restart.

SHOW ENGINE INNODB STATUS also shows the rate at which the read-ahead pages are read and the rate at which such pages are evicted without being accessed. The per-second averages are based on the statistics collected since the last invocation of SHOW ENGINE INNODB STATUS and are displayed in the BUFFER POOL AND MEMORY section of the SHOW ENGINE INNODB STATUS output.

For more information, see Section 17.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

### innodb_read_io_threads

| | |
|---|---|
| **Command-Line Format** | `--innodb-read-io-threads=#` |
| **System Variable** | `innodb_read_io_threads` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `(available logical processors / 2), min of 4` |
| **Minimum Value** | `1` |
| **Maximum Value** | `64` |

The number of I/O threads for read operations in `InnoDB`. Its counterpart for write threads is `innodb_write_io_threads`. For more information, see Section 17.8.5, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”. The default value is the number of available logical processors on the system divided by 2, with a minimum default value of 4.

Before MySQL 8.4, the default value was always 4.

> Note
>
> On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

`innodb_read_only`

| | |
|---|---|
| **Command-Line Format** | `--innodb-read-only[={OFF\|ON}]` |
| **System Variable** | `innodb_read_only` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Starts `InnoDB` in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. For more information, see Section 17.8.2, “Configuring InnoDB for Read-Only Operation”.

Enabling `innodb_read_only` prevents creating and dropping tables for all storage engines, and not only `InnoDB`. Table creation and drop operations for any storage engine modify data dictionary tables in the `mysql` system database, but those tables use the `InnoDB` storage engine and cannot be modified when `innodb_read_only` is enabled. The same principle applies to other table operations that require modifying data dictionary tables. Examples:

- If the `innodb_read_only` system variable is enabled, `ANALYZE TABLE` may fail because it cannot update statistics tables in the data dictionary, which use `InnoDB`. For `ANALYZE TABLE` operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a `MyISAM` table). To obtain the updated distribution statistics, set `information_schema_stats_expiry=0`.

- `ALTER TABLE tbl_name ENGINE=engine_name` fails because it updates the storage engine designation, which is stored in the data dictionary.

In addition, other tables in the `mysql` system database use the `InnoDB` storage engine. Making those tables read-only results in restrictions on operations that modify them. Examples:

- Account-management statements such as `CREATE USER` and `GRANT` fail because the grant tables use `InnoDB`.

- The `INSTALL PLUGIN` and `UNINSTALL PLUGIN` plugin-management statements fail because the `mysql.plugin` system table uses `InnoDB`.

- The `CREATE FUNCTION` and `DROP FUNCTION` loadable function-management statements fail because the `mysql.func` system table uses `InnoDB`.

`innodb_redo_log_archive_dirs`

| | |
|---|---|
| **Command-Line Format** | `--innodb-redo-log-archive-dirs` |
| **System Variable** | `innodb_redo_log_archive_dirs` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | String |
| **Default Value** | `NULL` |

Defines labeled directories where redo log archive files can be created. You can define multiple labeled directories in a semicolon-separated list. For example:

```
innodb_redo_log_archive_dirs='label1:/backups1;label2:/backups2'
```

A label can be any string of characters, with the exception of colons (:), which are not permitted. An empty label is also permitted, but the colon (:) is still required in this case.

A path must be specified, and the directory must exist. The path can contain colons (':'), but semicolons (;) are not permitted.

### innodb_redo_log_capacity

| | |
|---|---|
| **Command-Line Format** | `--innodb-redo-log-capacity=#` |
| **System Variable** | `innodb_redo_log_capacity` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `104857600` |
| **Minimum Value** | `8388608` |
| **Maximum Value** | `549755813888` |
| **Unit** | bytes |

Defines the amount of disk space occupied by redo log files.

`innodb_redo_log_capacity` supercedes the `innodb_log_files_in_group` and `innodb_log_file_size` variables, which are both ignored if `innodb_redo_log_capacity` is defined.

If `innodb_redo_log_capacity` is not defined, and if neither `innodb_log_file_size` or `innodb_log_files_in_group` are defined, then the default `innodb_redo_log_capacity` value is used.

If `innodb_redo_log_capacity` is not defined, and if `innodb_log_file_size` and/or `innodb_log_files_in_group` is defined, then the InnoDB redo log capacity is calculated as *(innodb_log_files_in_group * innodb_log_file_size)*. This calculation does not modify the unused `innodb_redo_log_capacity` setting's value.

The `Innodb_redo_log_capacity_resized` server status variable indicates the total redo log capacity for all redo log files.

If `innodb_dedicated_server` is enabled, the `innodb_redo_log_capacity` value is automatically configured if it is not explicitly defined. For more information, see Section 17.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

For more information, see Section 17.6.5, “Redo Log”.

### innodb_redo_log_encrypt

| | |
|---|---|
| **Command-Line Format** | `--innodb-redo-log-encrypt[={OFF\|ON}]` |
| **System Variable** | `innodb_redo_log_encrypt` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Controls encryption of redo log data for tables encrypted using the `InnoDB` data-at-rest encryption feature. Encryption of redo log data is disabled by default. For more information, see Redo Log Encryption.

### innodb_replication_delay

| | |
|---|---|
| **Command-Line Format** | `--innodb-replication-delay=#` |
| **System Variable** | `innodb_replication_delay` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `0` |
| **Minimum Value** | `0` |
| **Maximum Value** | `4294967295` |
| **Unit** | milliseconds |

The replication thread delay in milliseconds on a replica server if `innodb_thread_concurrency` is reached.

### innodb_rollback_on_timeout

| | |
|---|---|
| **Command-Line Format** | `--innodb-rollback-on-timeout[={OFF\|ON}]` |
| **System Variable** | `innodb_rollback_on_timeout` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

InnoDB rolls back only the last statement on a transaction timeout by default. If --innodb-rollback-on-timeout is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction.

For more information, see Section 17.20.5, “InnoDB Error Handling”.

### innodb_rollback_segments

| | |
|---|---|
| **Command-Line Format** | --innodb-rollback-segments=# |
| **System Variable** | innodb_rollback_segments |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 128 |
| **Minimum Value** | 1 |
| **Maximum Value** | 128 |

innodb_rollback_segments defines the number of rollback segments allocated to each undo tablespace and the global temporary tablespace for transactions that generate undo records. The number of transactions that each rollback segment supports depends on the InnoDB page size and the number of undo logs assigned to each transaction. For more information, see Section 17.6.6, “Undo Logs”.

For related information, see Section 17.3, “InnoDB Multi-Versioning”. For information about undo tablespaces, see Section 17.6.3.4, “Undo Tablespaces”.

### innodb_saved_page_number_debug

| | |
|---|---|
| **Command-Line Format** | --innodb-saved-page-number-debug=# |
| **System Variable** | innodb_saved_page_number_debug |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 2**32-1 |

Saves a page number. Setting the innodb_fil_make_page_dirty_debug option dirties the page defined by innodb_saved_page_number_debug. The innodb_saved_page_number_debug option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

### innodb_segment_reserve_factor

| | |
|---|---|
| **Command-Line Format** | --innodb-segment-reserve-factor=# |
| **System Variable** | innodb_segment_reserve_factor |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Numeric |
| **Default Value** | 12.5 |
| **Minimum Value** | 0.03 |
| **Maximum Value** | 40 |

Defines the percentage of tablespace file segment pages reserved as empty pages. The setting is applicable to file-per-table and general tablespaces. The innodb_segment_reserve_factor default setting is 12.5 percent, which is the same percentage of pages reserved in previous MySQL releases.

For more information, see Configuring the Percentage of Reserved File Segment Pages.

### innodb_sort_buffer_size

| | |
|---|---|
| **Command-Line Format** | --innodb-sort-buffer-size=# |
| **System Variable** | innodb_sort_buffer_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 1048576 |
| **Minimum Value** | 65536 |
| **Maximum Value** | 67108864 |
| **Unit** | bytes |

This variable defines the amount by which the temporary log file is extended when recording concurrent DML during an online DDL operation, and the size of the temporary log file read buffer and write buffer.

For more information, see Section 17.12.3, “Online DDL Space Requirements”.

`innodb_spin_wait_delay`

| | |
|---|---|
| **Command-Line Format** | `--innodb-spin-wait-delay=#` |
| **System Variable** | `innodb_spin_wait_delay` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `6` |
| **Minimum Value** | `0` |
| **Maximum Value** | `1000` |

The maximum delay between polls for a spin lock. The low-level implementation of this mechanism varies depending on the combination of hardware and operating system, so the delay does not correspond to a fixed time interval.

Can be used in combination with the `innodb_spin_wait_pause_multiplier` variable for greater control over the duration of spin-lock polling delays.

For more information, see Section 17.8.8, “Configuring Spin Lock Polling”.

`innodb_spin_wait_pause_multiplier`

| | |
|---|---|
| **Command-Line Format** | `--innodb-spin-wait-pause-multiplier=#` |
| **System Variable** | `innodb_spin_wait_pause_multiplier` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `50` |
| **Minimum Value** | `0` |
| **Maximum Value** | `100` |

Defines a multiplier value used to determine the number of PAUSE instructions in spin-wait loops that occur when a thread waits to acquire a mutex or rw-lock.

For more information, see Section 17.8.8, “Configuring Spin Lock Polling”.

`innodb_stats_auto_recalc`

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-auto-recalc[={OFF\|ON}]` |
| **System Variable** | `innodb_stats_auto_recalc` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

Causes `InnoDB` to automatically recalculate persistent statistics after the data in a table is changed substantially. The threshold value is 10% of the rows in the table. This setting applies to tables created when the `innodb_stats_persistent` option is enabled. Automatic statistics recalculation may also be configured by specifying `STATS_AUTO_RECALC=1` in a `CREATE TABLE` or `ALTER TABLE` statement. The amount of data sampled to produce the statistics is controlled by the `innodb_stats_persistent_sample_pages` variable.

For more information, see Section 17.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

`innodb_stats_include_delete_marked`

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-include-delete-marked[={OFF\|ON}]` |
| **System Variable** | `innodb_stats_include_delete_marked` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

By default, `InnoDB` reads uncommitted data when calculating statistics. In the case of an uncommitted transaction that deletes rows from a table, `InnoDB` excludes records that are delete-marked when calculating row estimates and index statistics, which can lead to non-optimal execution plans for other transactions that are operating on the table concurrently using a transaction isolation level other than `READ UNCOMMITTED`. To avoid this scenario, `innodb_stats_include_delete_marked` can be enabled to ensure that `InnoDB` includes delete-marked records when calculating persistent optimizer statistics.

When `innodb_stats_include_delete_marked` is enabled, `ANALYZE TABLE` considers delete-marked records when recalculating statistics.

`innodb_stats_include_delete_marked` is a global setting that affects all `InnoDB` tables. It is only applicable to persistent optimizer statistics.

For related information, see Section 17.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

### innodb_stats_method

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-method=value` |
| **System Variable** | `innodb_stats_method` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Enumeration |
| **Default Value** | `nulls_equal` |
| **Valid Values** | `nulls_equal`<br>`nulls_unequal`<br>`nulls_ignored` |

How the server treats `NULL` values when collecting statistics about the distribution of index values for `InnoDB` tables. Permitted values are `nulls_equal`, `nulls_unequal`, and `nulls_ignored`. For `nulls_equal`, all `NULL` index values are considered equal and form a single value group with a size equal to the number of `NULL` values. For `nulls_unequal`, `NULL` values are considered unequal, and each `NULL` forms a distinct value group of size 1. For `nulls_ignored`, `NULL` values are ignored.

The method used to generate table statistics influences how the optimizer chooses indexes for query execution, as described in Section 10.3.8, “InnoDB and MyISAM Index Statistics Collection”.

### innodb_stats_on_metadata

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-on-metadata[={OFF\|ON}]` |
| **System Variable** | `innodb_stats_on_metadata` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

This option only applies when optimizer statistics are configured to be non-persistent. Optimizer statistics are not persisted to disk when `innodb_stats_persistent` is disabled or when individual tables are created or altered with `STATS_PERSISTENT=0`. For more information, see Section 17.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

When `innodb_stats_on_metadata` is enabled, `InnoDB` updates non-persistent statistics when metadata statements such as `SHOW TABLE STATUS` or when accessing the Information Schema `TABLES` or `STATISTICS` tables. (These updates are similar to what happens for `ANALYZE TABLE`.) When disabled, `InnoDB` does not update statistics during these operations. Leaving the setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve `InnoDB` tables.

To change the setting, issue the statement `SET GLOBAL innodb_stats_on_metadata=`*`mode`*, where *`mode`* is either `ON` or `OFF` (or `1` or `0`). Changing the setting requires privileges sufficient to set global system variables (see Section 7.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

### innodb_stats_persistent

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-persistent[={OFF\|ON}]` |
| **System Variable** | `innodb_stats_persistent` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

Specifies whether `InnoDB` index statistics are persisted to disk. Otherwise, statistics may be recalculated frequently which can lead to variations in query execution plans. This setting is stored with each table when the table is created. You can set `innodb_stats_persistent` at the global level before creating a table, or use the `STATS_PERSISTENT` clause of the `CREATE TABLE` and `ALTER TABLE` statements to override the system-wide setting and configure persistent statistics for individual tables.

For more information, see Section 17.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

### innodb_stats_persistent_sample_pages

| | |
|---|---|
| **Command-Line Format** | `--innodb-stats-persistent-sample-pages=#` |
| **System Variable** | `innodb_stats_persistent_sample_pages` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |

| | |
|---|---|
| **Default Value** | 20 |
| **Minimum Value** | 1 |
| **Maximum Value** | 18446744073709551615 |

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of ANALYZE TABLE for an InnoDB table. For more information, see Section 17.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

> Note
>
> Setting a high value for innodb_stats_persistent_sample_pages could result in lengthy ANALYZE TABLE execution time. To estimate the number of database pages accessed by ANALYZE TABLE, see Section 17.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

innodb_stats_persistent_sample_pages only applies when innodb_stats_persistent is enabled for a table; when innodb_stats_persistent is disabled, innodb_stats_transient_sample_pages applies instead.

### innodb_stats_transient_sample_pages

| | |
|---|---|
| **Command-Line Format** | --innodb-stats-transient-sample-pages=# |
| **System Variable** | innodb_stats_transient_sample_pages |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 8 |
| **Minimum Value** | 1 |
| **Maximum Value** | 18446744073709551615 |

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an InnoDB table or recalculating statistics. For more information, see Section 17.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

> Note
>
> Setting a high value for innodb_stats_transient_sample_pages could result in lengthy ANALYZE TABLE execution time. To estimate the number of database pages accessed by ANALYZE TABLE, see Section 17.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

innodb_stats_transient_sample_pages only applies when innodb_stats_persistent is disabled for a table; when innodb_stats_persistent is enabled, innodb_stats_persistent_sample_pages applies instead. Takes the place of innodb_stats_sample_pages that was removed in MySQL 8.0. For more information, see Section 17.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

### innodb_status_output

| | |
|---|---|
| **Command-Line Format** | --innodb-status-output[={OFF\|ON}] |
| **System Variable** | innodb_status_output |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Enables or disables periodic output for the standard InnoDB Monitor. Also used in combination with innodb_status_output_locks to enable or disable periodic output for the InnoDB Lock Monitor. For more information, see Section 17.17.2, “Enabling InnoDB Monitors”.

### innodb_status_output_locks

| | |
|---|---|
| **Command-Line Format** | --innodb-status-output-locks[={OFF\|ON}] |
| **System Variable** | innodb_status_output_locks |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Enables or disables the InnoDB Lock Monitor. When enabled, the InnoDB Lock Monitor prints additional information about locks in SHOW ENGINE INNODB STATUS output and in periodic output printed to the MySQL error log. Periodic output for the InnoDB Lock Monitor is printed as part of the standard InnoDB Monitor output. The standard InnoDB Monitor must therefore be enabled for the InnoDB Lock Monitor to print data to the MySQL error log periodically. For more information, see Section 17.17.2, “Enabling InnoDB Monitors”.

### innodb_strict_mode

| | |
|---|---|
| **Command-Line Format** | `--innodb-strict-mode[={OFF\|ON}]` |
| **System Variable** | innodb_strict_mode |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

When innodb_strict_mode is enabled, `InnoDB` returns errors rather than warnings when checking for invalid or incompatible table options.

It checks that `KEY_BLOCK_SIZE`, `ROW_FORMAT`, `DATA DIRECTORY`, `TEMPORARY`, and `TABLESPACE` options are compatible with each other and other settings.

`innodb_strict_mode=ON` also enables a row size check when creating or altering a table, to prevent `INSERT` or `UPDATE` from failing due to the record being too large for the selected page size.

You can enable or disable innodb_strict_mode on the command line when starting `mysqld`, or in a MySQL configuration file. You can also enable or disable innodb_strict_mode at runtime with the statement `SET [GLOBAL|SESSION] innodb_strict_mode=`*`mode`*, where *`mode`* is either `ON` or `OFF`. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 7.1.9.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for innodb_strict_mode, and the setting affects only that client.

Setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 7.1.9.1, “System Variable Privileges”.

### innodb_sync_array_size

| | |
|---|---|
| **Command-Line Format** | `--innodb-sync-array-size=#` |
| **System Variable** | innodb_sync_array_size |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `1` |
| **Minimum Value** | `1` |
| **Maximum Value** | `1024` |

Defines the size of the mutex/lock wait array. Increasing the value splits the internal data structure used to coordinate threads, for higher concurrency in workloads with large numbers of waiting threads. This setting must be configured when the MySQL instance is starting up, and cannot be changed afterward. Increasing the value is recommended for workloads that frequently produce a large number of waiting threads, typically greater than 768.

### innodb_sync_spin_loops

| | |
|---|---|
| **Command-Line Format** | `--innodb-sync-spin-loops=#` |
| **System Variable** | innodb_sync_spin_loops |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | `30` |
| **Minimum Value** | `0` |
| **Maximum Value** | `4294967295` |

The number of times a thread waits for an `InnoDB` mutex to be freed before the thread is suspended.

### innodb_sync_debug

| | |
|---|---|
| **Command-Line Format** | `--innodb-sync-debug[={OFF\|ON}]` |
| **System Variable** | innodb_sync_debug |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Enables sync debug checking for the `InnoDB` storage engine. This option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

### innodb_table_locks

| | |
|---|---|
| **Command-Line Format** | `--innodb-table-locks[={OFF\|ON}]` |

| | |
|---|---|
| **System Variable** | `innodb_table_locks` |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

If `autocommit = 0`, `InnoDB` honors `LOCK TABLES`; MySQL does not return from `LOCK TABLES ... WRITE` until all other threads have released all their locks to the table. The default value of `innodb_table_locks` is 1, which means that `LOCK TABLES` causes InnoDB to lock a table internally if `autocommit = 0`.

`innodb_table_locks = 0` has no effect for tables locked explicitly with `LOCK TABLES ... WRITE`. It does have an effect for tables locked for read or write by `LOCK TABLES ... WRITE` implicitly (for example, through triggers) or by `LOCK TABLES ... READ`.

For related information, see Section 17.7, “InnoDB Locking and Transaction Model”.

`innodb_temp_data_file_path`

| | |
|---|---|
| **Command-Line Format** | `--innodb-temp-data-file-path=file_name` |
| **System Variable** | `innodb_temp_data_file_path` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | String |
| **Default Value** | `ibtmp1:12M:autoextend` |

Defines the relative path, name, size, and attributes of global temporary tablespace data files. The global temporary tablespace stores rollback segments for changes made to user-created temporary tables.

If no value is specified for `innodb_temp_data_file_path`, the default behavior is to create a single auto-extending data file named `ibtmp1` in the `innodb_data_home_dir` directory. The initial file size is slightly larger than 12MB.

The syntax for a global temporary tablespace data file specification includes the file name, file size, and `autoextend` and `max` attributes:

*`file_name`*`:`*`file_size`*`[:autoextend[:max:`*`max_file_size`*`]]`

The global temporary tablespace data file cannot have the same name as another `InnoDB` data file. Any inability or error creating the global temporary tablespace data file is treated as fatal and server startup is refused.

File sizes are specified in KB, MB, or GB by appending `K`, `M` or `G` to the size value. The sum of file sizes must be slightly larger than 12MB.

The size limit of individual files is determined by the operating system. File size can be more than 4GB on operating systems that support large files. Use of raw disk partitions for global temporary tablespace data files is not supported.

The `autoextend` and `max` attributes can be used only for the data file specified last in the `innodb_temp_data_file_path` setting. For example:

```
[mysqld]
innodb_temp_data_file_path=ibtmp1:50M;ibtmp2:12M:autoextend:max:500M
```

The `autoextend` option causes the data file to automatically increase in size when it runs out of free space. The `autoextend` increment is 64MB by default. To modify the increment, change the `innodb_autoextend_increment` variable setting.

The directory path for global temporary tablespace data files is formed by concatenating the paths defined by `innodb_data_home_dir` and `innodb_temp_data_file_path`.

Before running `InnoDB` in read-only mode, set `innodb_temp_data_file_path` to a location outside of the data directory. The path must be relative to the data directory. For example:

```
--innodb-temp-data-file-path=../../../tmp/ibtmp1:12M:autoextend
```

For more information, see Global Temporary Tablespace.

`innodb_temp_tablespaces_dir`

| | |
|---|---|
| **Command-Line Format** | `--innodb-temp-tablespaces-dir=dir_name` |
| **System Variable** | `innodb_temp_tablespaces_dir` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |
| **Default Value** | `#innodb_temp` |

Defines the location where InnoDB creates a pool of session temporary tablespaces at startup. The default location is the #innodb_temp directory in the data directory. A fully qualified path or path relative to the data directory is permitted.

Session temporary tablespaces always store user-created temporary tables and internal temporary tables created by the optimizer using InnoDB. (Previously, the on-disk storage engine for internal temporary tables was determined by the internal_tmp_disk_storage_engine system variable, which is no longer supported. See Storage Engine for On-Disk Internal Temporary Tables.)

For more information, see Session Temporary Tablespaces.

innodb_thread_concurrency

| | |
|---|---|
| **Command-Line Format** | --innodb-thread-concurrency=# |
| **System Variable** | innodb_thread_concurrency |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 1000 |

Defines the maximum number of threads permitted inside of InnoDB. A value of 0 (the default) is interpreted as infinite concurrency (no limit). This variable is intended for performance tuning on high concurrency systems.

InnoDB tries to keep the number of threads inside InnoDB less than or equal to the innodb_thread_concurrency limit. Threads waiting for locks are not counted in the number of concurrently executing threads.

The correct setting depends on workload and computing environment. Consider setting this variable if your MySQL instance shares CPU resources with other applications or if your workload or number of concurrent users is growing. Test a range of values to determine the setting that provides the best performance. innodb_thread_concurrency is a dynamic variable, which permits experimenting with different settings on a live test system. If a particular setting performs poorly, you can quickly set innodb_thread_concurrency back to 0.

Use the following guidelines to help find and maintain an appropriate setting:

- If the number of concurrent user threads for a workload is consistently small and does not affect performance, set innodb_thread_concurrency=0 (no limit).
- If your workload is consistently heavy or occasionally spikes, set an innodb_thread_concurrency value and adjust it until you find the number of threads that provides the best performance. For example, suppose that your system typically has 40 to 50 users, but periodically the number increases to 60, 70, or more. Through testing, you find that performance remains largely stable with a limit of 80 concurrent users. In this case, set innodb_thread_concurrency to 80.
- If you do not want InnoDB to use more than a certain number of virtual CPUs for user threads (20 virtual CPUs, for example), set innodb_thread_concurrency to this number (or possibly lower, depending on performance testing). If your goal is to isolate MySQL from other applications, consider binding the mysqld process exclusively to the virtual CPUs. Be aware, however, that exclusive binding can result in non-optimal hardware usage if the mysqld process is not consistently busy. In this case, you can bind the mysqld process to the virtual CPUs but allow other applications to use some or all of the virtual CPUs.

  Note

  From an operating system perspective, using a resource management solution to manage how CPU time is shared among applications may be preferable to binding the mysqld process. For example, you could assign 90% of virtual CPU time to a given application while other critical processes *are not* running, and scale that value back to 40% when other critical processes *are* running.
- In some cases, the optimal innodb_thread_concurrency setting can be smaller than the number of virtual CPUs.
- An innodb_thread_concurrency value that is too high can cause performance regression due to increased contention on system internals and resources.
- Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the innodb_thread_concurrency setting.

A value of 0 disables the queries inside InnoDB and queries in queue counters in the ROW OPERATIONS section of SHOW ENGINE INNODB STATUS output.

For related information, see Section 17.8.4, “Configuring Thread Concurrency for InnoDB”.

innodb_thread_sleep_delay

| | |
|---|---|
| **Command-Line Format** | --innodb-thread-sleep-delay=# |
| **System Variable** | innodb_thread_sleep_delay |
| **Scope** | Global |

| | |
|---|---|
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 10000 |
| **Minimum Value** | 0 |
| **Maximum Value** | 1000000 |
| **Unit** | microseconds |

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10000. A value of 0 disables sleep. You can set innodb_adaptive_max_sleep_delay to the highest value you would allow for innodb_thread_sleep_delay, and InnoDB automatically adjusts innodb_thread_sleep_delay up or down depending on current thread-scheduling activity. This dynamic adjustment helps the thread scheduling mechanism to work smoothly during times when the system is lightly loaded or when it is operating near full capacity.

For more information, see Section 17.8.4, “Configuring Thread Concurrency for InnoDB”.

innodb_tmpdir

| | |
|---|---|
| **Command-Line Format** | --innodb-tmpdir=dir_name |
| **System Variable** | innodb_tmpdir |
| **Scope** | Global, Session |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |
| **Default Value** | NULL |

Used to define an alternate directory for temporary sort files created during online ALTER TABLE operations that rebuild the table.

Online ALTER TABLE operations that rebuild the table also create an *intermediate* table file in the same directory as the original table. The innodb_tmpdir option is not applicable to intermediate table files.

A valid value is any directory path other than the MySQL data directory path. If the value is NULL (the default), temporary files are created MySQL temporary directory ($TMPDIR on Unix, %TEMP% on Windows, or the directory specified by the --tmpdir configuration option). If a directory is specified, existence of the directory and permissions are only checked when innodb_tmpdir is configured using a SET statement. If a symlink is provided in a directory string, the symlink is resolved and stored as an absolute path. The path should not exceed 512 bytes. An online ALTER TABLE operation reports an error if innodb_tmpdir is set to an invalid directory. innodb_tmpdir overrides the MySQL tmpdir setting but only for online ALTER TABLE operations.

The FILE privilege is required to configure innodb_tmpdir.

The innodb_tmpdir option was introduced to help avoid overflowing a temporary file directory located on a tmpfs file system. Such overflows could occur as a result of large temporary sort files created during online ALTER TABLE operations that rebuild the table.

In replication environments, only consider replicating the innodb_tmpdir setting if all servers have the same operating system environment. Otherwise, replicating the innodb_tmpdir setting could result in a replication failure when running online ALTER TABLE operations that rebuild the table. If server operating environments differ, it is recommended that you configure innodb_tmpdir on each server individually.

For more information, see Section 17.12.3, “Online DDL Space Requirements”. For information about online ALTER TABLE operations, see Section 17.12, “InnoDB and Online DDL”.

innodb_trx_purge_view_update_only_debug

| | |
|---|---|
| **Command-Line Format** | --innodb-trx-purge-view-update-only-debug[={OFF\|ON}] |
| **System Variable** | innodb_trx_purge_view_update_only_debug |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | OFF |

Pauses purging of delete-marked records while allowing the purge view to be updated. This option artificially creates a situation in which the purge view is updated but purges have not yet been performed. This option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

innodb_trx_rseg_n_slots_debug

| | |
|---|---|
| **Command-Line Format** | --innodb-trx-rseg-n-slots-debug=# |
| **System Variable** | innodb_trx_rseg_n_slots_debug |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 0 |
| **Minimum Value** | 0 |
| **Maximum Value** | 1024 |

Sets a debug flag that limits `TRX_RSEG_N_SLOTS` to a given value for the `trx_rsegf_undo_find_free` function that looks for free slots for undo log segments. This option is only available if debugging support is compiled in using the `WITH_DEBUG` **CMake** option.

`innodb_undo_directory`

| | |
|---|---|
| **Command-Line Format** | `--innodb-undo-directory=dir_name` |
| **System Variable** | `innodb_undo_directory` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Directory name |

The path where `InnoDB` creates undo tablespaces. Typically used to place undo tablespaces on a different storage device.

There is no default value (it is NULL). If the `innodb_undo_directory` variable is undefined, undo tablespaces are created in the data directory.

The default undo tablespaces (`innodb_undo_001` and `innodb_undo_002`) created when the MySQL instance is initialized always reside in the directory defined by the `innodb_undo_directory` variable.

Undo tablespaces created using `CREATE UNDO TABLESPACE` syntax are created in the directory defined by the `innodb_undo_directory` variable if a different path is not specified.

For more information, see Section 17.6.3.4, “Undo Tablespaces”.

`innodb_undo_log_encrypt`

| | |
|---|---|
| **Command-Line Format** | `--innodb-undo-log-encrypt[={OFF\|ON}]` |
| **System Variable** | `innodb_undo_log_encrypt` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `OFF` |

Controls encryption of undo log data for tables encrypted using the `InnoDB` data-at-rest encryption feature. Only applies to undo logs that reside in separate undo tablespaces. See Section 17.6.3.4, “Undo Tablespaces”. Encryption is not supported for undo log data that resides in the system tablespace. For more information, see Undo Log Encryption.

`innodb_undo_log_truncate`

| | |
|---|---|
| **Command-Line Format** | `--innodb-undo-log-truncate[={OFF\|ON}]` |
| **System Variable** | `innodb_undo_log_truncate` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | `ON` |

When enabled, undo tablespaces that exceed the threshold value defined by `innodb_max_undo_log_size` are marked for truncation. Only undo tablespaces can be truncated. Truncating undo logs that reside in the system tablespace is not supported. For truncation to occur, there must be at least two undo tablespaces.

The `innodb_purge_rseg_truncate_frequency` variable can be used to expedite truncation of undo tablespaces.

For more information, see Truncating Undo Tablespaces.

`innodb_undo_tablespaces`

| | |
|---|---|
| **Command-Line Format** | `--innodb-undo-tablespaces=#` |
| **Deprecated** | Yes |
| **System Variable** | `innodb_undo_tablespaces` |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 2 |
| **Minimum Value** | 2 |
| **Maximum Value** | 127 |

Defines the number of undo tablespaces used by `InnoDB`. The default and minimum value is 2.

Note

The innodb_undo_tablespaces variable is deprecated; setting it has no effect. You should expect it to be removed in a future MySQL release.

For more information, see Section 17.6.3.4, “Undo Tablespaces”.

innodb_use_fdatasync

| | |
|---|---|
| **Command-Line Format** | --innodb-use-fdatasync[={OFF\|ON}] |
| **System Variable** | innodb_use_fdatasync |
| **Scope** | Global |
| **Dynamic** | Yes |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

On platforms that support fdatasync() system calls, having innodb_use_fdatasync enabled permits using fdatasync() instead of fsync() system calls for operating system flushes. An fdatasync() call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

A subset of innodb_flush_method settings such as fsync, O_DSYNC, and O_DIRECT use fsync() system calls. The innodb_use_fdatasync variable is applicable when using those settings.

Before MySQL 8.4, this option was disabled by default.

innodb_use_native_aio

| | |
|---|---|
| **Command-Line Format** | --innodb-use-native-aio[={OFF\|ON}] |
| **System Variable** | innodb_use_native_aio |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Specifies whether to use the asynchronous I/O subsystem. This variable cannot be changed while the server is running. Normally, you do not need to configure this option, because it is enabled by default.

This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in SHOW ENGINE INNODB STATUS output.

Running with a large number of InnoDB I/O threads, and especially running multiple such instances on the same server machine, can exceed capacity limits on Linux systems. In this case, you may receive the following error:

```
EAGAIN: The specified maxevents exceeds the user's limit of available events.
```

You can typically address this error by writing a higher limit to /proc/sys/fs/aio-max-nr.

However, if a problem with the asynchronous I/O subsystem in the OS prevents InnoDB from starting, you can start the server with innodb_use_native_aio=0. This option may also be disabled automatically during startup if InnoDB detects a potential problem such as a combination of tmpdir location, tmpfs file system, and Linux kernel that does not support AIO on tmpfs.

For more information, see Section 17.8.6, “Using Asynchronous I/O on Linux”.

innodb_validate_tablespace_paths

| | |
|---|---|
| **Command-Line Format** | --innodb-validate-tablespace-paths[={OFF\|ON}] |
| **System Variable** | innodb_validate_tablespace_paths |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Boolean |
| **Default Value** | ON |

Controls tablespace file path validation. At startup, InnoDB validates the paths of known tablespace files against tablespace file paths stored in the data dictionary in case tablespace files have been moved to a different location. The innodb_validate_tablespace_paths variable permits disabling tablespace path validation. This feature is intended for environments where tablespaces files are not moved. Disabling path validation improves startup time on systems with a large number of tablespace files.

Warning

Starting the server with tablespace path validation disabled after moving tablespace files can lead to undefined behavior.

For more information, see Section 17.6.3.7, “Disabling Tablespace Path Validation”.

`innodb_version`

The `InnoDB` version number. This is a legacy variable, the value is the same as the MySQL server `version`.

`innodb_write_io_threads`

| | |
|---|---|
| **Command-Line Format** | `--innodb-write-io-threads=#` |
| **System Variable** | `innodb_write_io_threads` |
| **Scope** | Global |
| **Dynamic** | No |
| **SET_VAR Hint Applies** | No |
| **Type** | Integer |
| **Default Value** | 4 |
| **Minimum Value** | 1 |
| **Maximum Value** | 64 |

The number of I/O threads for write operations in `InnoDB`. The default value is 4. Its counterpart for read threads is `innodb_read_io_threads`. For more information, see Section 17.8.5, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.

> Note
>
> On Linux systems, running multiple MySQL servers (typically more than 12) with default settings for `innodb_read_io_threads`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

Also take into consideration the value of `sync_binlog`, which controls synchronization of the binary log to disk.

For general I/O tuning advice, see Section 10.5.8, “Optimizing InnoDB Disk I/O”.