14.15 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.
- 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 5.1.8.2, “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 14.8.1, “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 your database workload.
- For more information on specifying options and system variables, see Section 4.2.2, “Specifying Program Options”.

Table 14.18 InnoDB Option and Variable Reference

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
daemon_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
foreign_key_checks			Yes		Both	Yes
ignore_builtin_innodb	Yes	Yes	Yes		Global	No
innodb	Yes	Yes				

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
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_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
Innodb_available_undo_logs				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_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_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

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
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	Varies
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_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_checksums	Yes	Yes	Yes		Global	No
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
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	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

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
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_deadlock_detect	Yes	Yes	Yes		Global	Yes
innodb_default_row_format	Yes	Yes	Yes		Global	Yes
innodb_disable_resize_buffer_pool_debug	Yes	Yes	Yes		Global	Yes
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	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_format	Yes	Yes	Yes		Global	Yes
innodb_file_format_check	Yes	Yes	Yes		Global	No
innodb_file_format_max	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

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
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_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_large_prefix	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_locks_unsafe_for_binlog	Yes	Yes	Yes		Global	No
innodb_log_buffer_size	Yes	Yes	Yes		Global	No
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_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
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	No
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

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
innodb_print_all_deadlocks	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_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_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	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

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
innodb_stats_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_support_xa	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_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	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_truncate	Yes	Yes	Yes		Global	Yes
innodb_undo_logs	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	No
innodb_use_native_aio	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[=value]`

Command-Line Format	`--innodb[=value]`
Deprecated	Yes
Type	Enumeration
Default Value	`ON`

Valid Values	`OFF`
	`ON`
	`FORCE`

Controls loading of the `InnoDB` storage engine, if the server was compiled with `InnoDB` support. This option has a tristate format, with possible values of `OFF`, `ON`, or `FORCE`. See Section 5.5.1, “Installing and Uninstalling Plugins”.

To disable `InnoDB`, use `--innodb=OFF` or `--skip-innodb`. In this case, because the default storage engine is `InnoDB`, the server does not start unless you also use `--default-storage-engine` and `--default-tmp-storage-engine` to set the default to some other engine for both permanent and `TEMPORARY` tables.

The `InnoDB` storage engine can no longer be disabled, and the `--innodb=OFF` and `--skip-innodb` options are deprecated and have no effect. Their use results in a warning. You should expect these options to be removed in a future MySQL release.

- `--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 14.18.2, “Enabling InnoDB Monitors”.

- `--skip-innodb`

  Disable the `InnoDB` storage engine. See the description of `--innodb`.

InnoDB System Variables

- `daemon_memcached_enable_binlog`

Command-Line Format	`--daemon-memcached-enable-binlog[={OFF|ON}]`
System Variable	`daemon_memcached_enable_binlog`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

  Enable this option on the source server to use the `InnoDB` **memcached** plugin (`daemon_memcached`) with the MySQL binary log. This option can only be set at server startup. You must also enable the MySQL binary log on the source server using the `--log-bin` option.

  For more information, see Section 14.21.6, “The InnoDB memcached Plugin and Replication”.

- `daemon_memcached_engine_lib_name`

Command-Line Format	`--daemon-memcached-engine-lib-name=file_name`
System Variable	`daemon_memcached_engine_lib_name`
Scope	Global
Dynamic	No
Type	File name
Default Value	`innodb_engine.so`

  Specifies the shared library that implements the `InnoDB` **memcached** plugin.

  For more information, see Section 14.21.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_engine_lib_path`

Command-Line Format	`--daemon-memcached-engine-lib-path=dir_name`
System Variable	`daemon_memcached_engine_lib_path`
Scope	Global

Dynamic	No
Type	Directory name
Default Value	`NULL`

The path of the directory containing the shared library that implements the `InnoDB` **memcached** plugin. The default value is NULL, representing the MySQL plugin directory. You should not need to modify this parameter unless specifying a `memcached` plugin for a different storage engine that is located outside of the MySQL plugin directory.

For more information, see Section 14.21.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_option`

Command-Line Format	`--daemon-memcached-option=options`
System Variable	`daemon_memcached_option`
Scope	Global
Dynamic	No
Type	String
Default Value	

Used to pass space-separated memcached options to the underlying **memcached** memory object caching daemon on startup. For example, you might change the port that **memcached** listens on, reduce the maximum number of simultaneous connections, change the maximum memory size for a key-value pair, or enable debugging messages for the error log.

See Section 14.21.3, “Setting Up the InnoDB memcached Plugin” for usage details. For information about **memcached** options, refer to the **memcached** man page.

- `daemon_memcached_r_batch_size`

Command-Line Format	`--daemon-memcached-r-batch-size=#`
System Variable	`daemon_memcached_r_batch_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`
Minimum Value	`1`

Maximum Value	1073741824

Specifies how many **memcached** read operations (`get` operations) to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_w_batch_size`.

This value is set to 1 by default, so that any changes made to the table through SQL statements are immediately visible to **memcached** operations. You might increase it to reduce the overhead from frequent commits on a system where the underlying table is only being accessed through the **memcached** interface. If you set the value too large, the amount of undo or redo data could impose some storage overhead, as with any long-running transaction.

For more information, see Section 14.21.3, “Setting Up the InnoDB memcached Plugin”.

- `daemon_memcached_w_batch_size`

Command-Line Format	`--daemon-memcached-w-batch-size=#`
System Variable	`daemon_memcached_w_batch_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`1048576`

Specifies how many **memcached** write operations, such as `add`, `set`, and `incr`, to perform before doing a `COMMIT` to start a new transaction. Counterpart of `daemon_memcached_r_batch_size`.

This value is set to 1 by default, on the assumption that data being stored is important to preserve in case of an outage and should immediately be committed. When storing non-critical data, you might increase this value to reduce the overhead from frequent commits; but then the last ***N***-1 uncommitted write operations could be lost if an unexpected exit occurs.

For more information, see Section 14.21.3, “Setting Up the InnoDB memcached Plugin”.

- `ignore_builtin_innodb`

Command-Line Format	`--ignore-builtin-innodb[={OFF|ON}]`
Deprecated	Yes
System Variable	`ignore_builtin_innodb`

Scope	Global
Dynamic	No
Type	Boolean

In earlier versions of MySQL, enabling this variable caused the server to behave as if the built-in `InnoDB` were not present, which enabled the `InnoDB Plugin` to be used instead. In MySQL 5.7, `InnoDB` is the default storage engine and `InnoDB Plugin` is not used, so this variable is ignored.

- `innodb_adaptive_flushing`

Command-Line Format	`--innodb-adaptive-flushing[={OFF|ON}]`
System Variable	`innodb_adaptive_flushing`
Scope	Global
Dynamic	Yes
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 14.8.3.5, “Configuring Buffer Pool Flushing” for more information. For general I/O tuning advice, see Section 8.5.8, “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
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 14.8.3.5, “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
Type	Boolean
Default Value	`ON`

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 14.5.3, “Adaptive Hash Index” for details.

This variable is enabled 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 5.1.8.1, “System Variable Privileges”. You can also use `--skip-innodb-adaptive-hash-index` at server startup to disable 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.

- `innodb_adaptive_hash_index_parts`

Command-Line Format	`--innodb-adaptive-hash-index-parts=#`
System Variable	`innodb_adaptive_hash_index_parts`
Scope	Global
Dynamic	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.

In earlier releases, the adaptive hash index search system was protected by a single latch (`btr_search_latch`) which could become a point of contention. With the introduction of the `innodb_adaptive_hash_index_parts` option, the search system is partitioned into 8 parts by default. The maximum setting is 512.

For related information, see Section 14.5.3, “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
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 14.8.5, “Configuring Thread Concurrency for InnoDB”.

- `innodb_api_bk_commit_interval`

Command-Line Format	`--innodb-api-bk-commit-interval=#`
System Variable	`innodb_api_bk_commit_interval`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`5`
Minimum Value	`1`

Maximum Value	1073741824
Unit	seconds

How often to auto-commit idle connections that use the InnoDB **memcached** interface, in seconds. For more information, see Section 14.21.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_disable_rowlock

Command-Line Format	--innodb-api-disable-rowlock[={OFF|ON}]
System Variable	innodb_api_disable_rowlock
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

Use this option to disable row locks when InnoDB **memcached** performs DML operations. By default, innodb_api_disable_rowlock is disabled, which means that **memcached** requests row locks for get and set operations. When innodb_api_disable_rowlock is enabled, **memcached** requests a table lock instead of row locks.

innodb_api_disable_rowlock is not dynamic. It must be specified on the **mysqld** command line or entered in the MySQL configuration file. Configuration takes effect when the plugin is installed, which occurs when the MySQL server is started.

For more information, see Section 14.21.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- innodb_api_enable_binlog

Command-Line Format	--innodb-api-enable-binlog[={OFF|ON}]
System Variable	innodb_api_enable_binlog
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

Lets you use the `InnoDB` **memcached** plugin with the MySQL binary log. For more information, see Enabling the InnoDB memcached Binary Log.

- `innodb_api_enable_mdl`

Command-Line Format	`--innodb-api-enable-mdl[={OFF|ON}]`
System Variable	`innodb_api_enable_mdl`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`OFF`

Locks the table used by the `InnoDB` **memcached** plugin, so that it cannot be dropped or altered by DDL through the SQL interface. For more information, see Section 14.21.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- `innodb_api_trx_level`

Command-Line Format	`--innodb-api-trx-level=#`
System Variable	`innodb_api_trx_level`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`3`

Controls the transaction isolation level on queries processed by the **memcached** interface. The constants corresponding to the familiar names are:

- 0 = `READ UNCOMMITTED`
- 1 = `READ COMMITTED`
- 2 = `REPEATABLE READ`
- 3 = `SERIALIZABLE`

For more information, see Section 14.21.5.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin”.

- `innodb_autoextend_increment`

Command-Line Format	`--innodb-autoextend-increment=#`
System Variable	`innodb_autoextend_increment`
Scope	Global
Dynamic	Yes
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
Type	Integer
Default Value	`1`

Valid Values	`0`
	`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 1 (consecutive). 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}]`
Introduced	5.7.10
System Variable	`innodb_background_drop_list_empty`
Scope	Global
Dynamic	Yes
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
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 the effect it has 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 14.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

- `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
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 14.8.3.6, “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
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.

For more information, see Section 14.8.3.6, “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
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.

The change to the innodb_buffer_pool_dump_pct default value coincides with default value changes for innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup, which are both enabled by default in MySQL 5.7.

- innodb_buffer_pool_filename

Command-Line Format	--innodb-buffer-pool-filename=file_name
System Variable	innodb_buffer_pool_filename
Scope	Global
Dynamic	Yes
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 14.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_instances

Command-Line Format	--innodb-buffer-pool-instances=#

System Variable	`innodb_buffer_pool_instances`
Scope	Global
Dynamic	No
Type	Integer
Default Value (Windows, 32-bit platforms)	`(autosized)`
Default Value (Other)	`8 (or 1 if innodb_buffer_pool_size < 1GB)`
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.

This option only takes effect when setting `innodb_buffer_pool_size` to 1GB or more. 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.

The default value on 32-bit Windows systems depends on the value of `innodb_buffer_pool_size`, as described below:

- If `innodb_buffer_pool_size` is greater than 1.3GB, the default for `innodb_buffer_pool_instances` is `innodb_buffer_pool_size`/128MB, with individual memory allocation requests for each chunk. 1.3GB was chosen as the boundary at which there is significant risk for 32-bit Windows to be unable to allocate the contiguous address space needed for a single buffer pool.
- Otherwise, the default is 1.

On all other platforms, the default value is 8 when `innodb_buffer_pool_size` is greater than or equal to 1GB. Otherwise, the default is 1.

For related information, see Section 14.8.3.1, “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
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 14.8.3.6, “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
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 14.8.3.6, “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

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 14.8.3.6, “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
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 14.8.3.6, “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 14.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

- `innodb_change_buffer_max_size`

Command-Line Format	`--innodb-change-buffer-max-size=#`
System Variable	`innodb_change_buffer_max_size`
Scope	Global
Dynamic	Yes
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 14.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_change_buffering

Command-Line Format	--innodb-change-buffering=value
System Variable	innodb_change_buffering
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	all
Valid Values	none inserts deletes changes purges 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.

Table 14.19 Permitted Values for innodb_change_buffering

Value	Description
none	Do not buffer any operations.
inserts	Buffer insert operations.
deletes	Buffer delete marking operations; strictly speaking, the writes that mark index records for later deletion during a purge operation.
changes	Buffer inserts and delete-marking operations.
purges	Buffer the physical deletion operations that happen in the background.
all	The default. Buffer inserts, delete-marking operations, and purges.

For more information, see Section 14.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “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
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_checksum_algorithm`

Command-Line Format	`--innodb-checksum-algorithm=value`
System Variable	`innodb_checksum_algorithm`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`crc32`

Valid Values	`crc32`
	`strict_crc32`
	`innodb`
	`strict_innodb`
	`none`
	`strict_none`

Specifies how to generate and verify the checksum stored in the disk blocks of `InnoDB` tablespaces. `crc32` is the default value as of MySQL 5.7.7.

`innodb_checksum_algorithm` replaces the `innodb_checksums` option. The following values were provided for compatibility, up to and including MySQL 5.7.6:

- `innodb_checksums=ON` is the same as `innodb_checksum_algorithm=innodb`.
- `innodb_checksums=OFF` is the same as `innodb_checksum_algorithm=none`.

As of MySQL 5.7.7, with a default `innodb_checksum_algorithm` value of crc32, `innodb_checksums=ON` is now the same as `innodb_checksum_algorithm=crc32`. `innodb_checksums=OFF` is still the same as `innodb_checksum_algorithm=none`.

To avoid conflicts, remove references to `innodb_checksums` from MySQL configuration files and startup scripts.

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.

> **Note**
>
> Prior to MySQL 5.7.8, a strict mode setting for `innodb_checksum_algorithm` caused `InnoDB` to halt when encountering a *valid* but non-matching checksum. In MySQL 5.7.8 and later, only an error message is printed, and the page is accepted as valid if it has a valid `innodb`, `crc32` or `none` checksum.

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 14.20 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.

Versions of MySQL Enterprise Backup up to 3.8.0 do not support backing up tablespaces that use CRC32 checksums. MySQL Enterprise Backup adds CRC32 checksum support in 3.8.1, with some limitations. Refer to the MySQL Enterprise Backup 3.8.1 Change History for more information.

- innodb_checksums

Command-Line Format	--innodb-checksums[={OFF|ON}]
Deprecated	Yes
System Variable	innodb_checksums
Scope	Global
Dynamic	No
Type	Boolean
Default Value	ON

InnoDB can use checksum validation on all tablespace pages read from disk to ensure extra fault tolerance against hardware faults or corrupted data files. This validation is enabled by default. Under specialized circumstances (such as when running benchmarks) this safety feature can be disabled with --skip-innodb-checksums. You can specify the method of calculating the checksum using the innodb_checksum_algorithm option.

innodb_checksums is deprecated, replaced by innodb_checksum_algorithm.

Prior to MySQL 5.7.7, innodb_checksums=ON is the same as innodb_checksum_algorithm=innodb. As of MySQL 5.7.7, the innodb_checksum_algorithm default value is crc32, and innodb_checksums=ON is the same as innodb_checksum_algorithm=crc32. innodb_checksums=OFF is the same as innodb_checksum_algorithm=none.

Remove any innodb_checksums options from your configuration files and startup scripts to avoid conflicts with innodb_checksum_algorithm. innodb_checksums=OFF automatically sets innodb_checksum_algorithm=none. innodb_checksums=ON is ignored and overridden by any other setting for innodb_checksum_algorithm.

- 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
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 24.4.7, “The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables”, and Section 14.9.1.4, “Monitoring InnoDB Table Compression at Runtime”.

- innodb_commit_concurrency

Command-Line Format	--innodb-commit-concurrency=#
System Variable	innodb_commit_concurrency

Scope	Global
Dynamic	Yes
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
Type	Enumeration
Default Value	`none`

Valid Values	`none`
	`zlib`
	`lz4`
	`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 14.9.2, “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

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 14.9.1.6, “Compression for OLTP Workloads”.

- innodb_compression_level

Command-Line Format	--innodb-compression-level=#
System Variable	innodb_compression_level
Scope	Global
Dynamic	Yes
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 14.9.1.6, “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	

Type	Yes 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 14.9.1.6, “Compression for OLTP Workloads”.

- `innodb_concurrency_tickets`

Command-Line Format	`--innodb-concurrency-tickets=#`
System Variable	`innodb_concurrency_tickets`
Scope	Global
Dynamic	Yes
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 14.8.5, “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
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
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. If you specify the value as an empty string, you can specify an absolute path for innodb_data_file_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 14.8.1, “InnoDB Startup Configuration”.

- innodb_deadlock_detect

Command-Line Format	--innodb-deadlock-detect[={OFF|ON}]
Introduced	5.7.15
System Variable	innodb_deadlock_detect
Scope	Global
Dynamic	Yes
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 14.7.5.2, “Deadlock Detection”.

- innodb_default_row_format

Command-Line Format	--innodb-default-row-format=value
System Variable	innodb_default_row_format
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	DYNAMIC
Valid Values	REDUNDANT COMPACT 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_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
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_disable_resize_buffer_pool_debug

Command-Line Format	--innodb-disable-resize-buffer-pool-debug[={OFF|ON}]
System Variable	innodb_disable_resize_buffer_pool_debug
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	ON

Disables resizing of the InnoDB buffer pool. This option is only available if debugging support is compiled in using the WITH_DEBUG **CMake** option.

- innodb_doublewrite

Command-Line Format	--innodb-doublewrite[={OFF|ON}]
System Variable	innodb_doublewrite
Scope	Global
Dynamic	No
Type	Boolean
Default Value	ON

When enabled (the default), InnoDB stores all data twice, first to the doublewrite buffer, then to the actual data files. This variable can be turned off with --skip-innodb-doublewrite for benchmarks or cases when top performance is needed rather than concern for data integrity or possible failures.

If system tablespace data files (ibdata* files) are located on Fusion-io devices that support atomic writes, doublewrite buffering is automatically disabled and Fusion-io atomic writes are used for all data files. Because the doublewrite buffer setting is global, doublewrite buffering is also disabled for data files residing on non-Fusion-io hardware. This feature is only supported on Fusion-io hardware and 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 14.6.5, “Doublewrite Buffer”.

- innodb_fast_shutdown

Command-Line Format	--innodb-fast-shutdown=#

System Variable	`innodb_fast_shutdown`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`1`

Valid Values	`0`
	`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
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_format

Command-Line Format	--innodb-file-format=value
Deprecated	Yes
System Variable	innodb_file_format
Scope	Global
Dynamic	Yes
Type	String
Default Value	Barracuda
Valid Values	Antelope Barracuda

Enables an InnoDB file format for file-per-table tablespaces. Supported file formats are Antelope and Barracuda. Antelope is the original InnoDB file format, which supports REDUNDANT and COMPACT row formats. Barracuda is the newer file format, which supports COMPRESSED and DYNAMIC row formats.

COMPRESSED and DYNAMIC row formats enable important storage features for InnoDB tables. See Section 14.11, “InnoDB Row Formats”.

Changing the innodb_file_format setting does not affect the file format of existing InnoDB tablespace files.

The innodb_file_format setting does not apply to general tablespaces, which support tables of all row formats. See Section 14.6.3.3, “General Tablespaces”.

The innodb_file_format default value was changed to Barracuda in MySQL 5.7.

The innodb_file_format setting is ignored when creating tables that use the DYNAMIC row format. A table created using the DYNAMIC row format always uses the Barracuda file format, regardless of the innodb_file_format setting. To use the COMPRESSED row format, innodb_file_format must be set to Barracuda.

The innodb_file_format option is deprecated; expect it to be removed in a future release. The purpose of the innodb_file_format option was to allow users to downgrade to the built-in version of InnoDB in earlier versions of MySQL. Now that those versions of MySQL have reached the end of their product lifecycles, downgrade support provided by this option is no longer necessary.

For more information, see Section 14.10, “InnoDB File-Format Management”.

- `innodb_file_format_check`

Command-Line Format	`--innodb-file-format-check[={OFF|ON}]`
Deprecated	Yes
System Variable	`innodb_file_format_check`
Scope	Global
Dynamic	No
Type	Boolean
Default Value	`ON`

This variable can be set to 1 or 0 at server startup to enable or disable whether `InnoDB` checks the file format tag in the system tablespace (for example, `Antelope` or `Barracuda`). If the tag is checked and is higher than that supported by the current version of `InnoDB`, an error occurs and `InnoDB` does not start. If the tag is not higher, `InnoDB` sets the value of `innodb_file_format_max` to the file format tag.

> **Note**
>
> Despite the default value sometimes being displayed as `ON` or `OFF`, always use the numeric values 1 or 0 to turn this option on or off in your configuration file or command line string.

For more information, see Section 14.10.2.1, “Compatibility Check When InnoDB Is Started”.

The `innodb_file_format_check` option is deprecated together with the `innodb_file_format` option. You should expect both options to be removed in a future release.

- `innodb_file_format_max`

Command-Line Format	`--innodb-file-format-max=value`
Deprecated	Yes
System Variable	`innodb_file_format_max`
Scope	Global
Dynamic	Yes
Type	String
Default Value	`Barracuda`
Valid Values	`Antelope`

	Barracuda

At server startup, InnoDB sets the value of this variable to the file format tag in the system tablespace (for example, Antelope or Barracuda). If the server creates or opens a table with a "higher" file format, it sets the value of innodb_file_format_max to that format.

For related information, see Section 14.10, "InnoDB File-Format Management".

The innodb_file_format_max option is deprecated together with the innodb_file_format option. You should expect both options to be removed in a future release.

- innodb_file_per_table

Command-Line Format	--innodb-file-per-table[={OFF|ON}]
System Variable	innodb_file_per_table
Scope	Global
Dynamic	Yes
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 14.6.3.2, "File-Per-Table Tablespaces". For information about the InnoDB system tablespace, see Section 14.6.3.1, "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 5.1.8.1, "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.

When innodb_file_per_table is enabled, a table-copying ALTER TABLE operation on a table that resides in the system tablespace implicitly re-creates the table in a file-per-table tablespace. To prevent this from occurring, disable innodb_file_per_table before executing table-copying ALTER TABLE operations on tables that reside in the system tablespace.

The innodb_file_per-table setting does not affect the creation of temporary tables. Temporary tables are created in the temporary tablespace. See Section 14.6.3.5, “The Temporary Tablespace”.

- innodb_fill_factor

Command-Line Format	--innodb-fill-factor=#
System Variable	innodb_fill_factor
Scope	Global
Dynamic	Yes
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 14.6.2.3, “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
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
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`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 16.3.2, “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
Type	String
Default Value	`NULL`

Valid Values (Unix)	`fsync`
	`O_DSYNC`
	`littlesync`

	nosync O_DIRECT O_DIRECT_NO_FSYNC
Valid Values (Windows)	async_unbuffered normal unbuffered

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

If innodb_flush_method is set to NULL on a Unix-like system, the fsync option is used by default. If innodb_flush_method is set to NULL on Windows, the async_unbuffered option is used by default.

The innodb_flush_method options for Unix-like systems include:

- fsync: InnoDB uses the fsync() system call to flush both the data and log files. fsync is the default setting.
- O_DSYNC: 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: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- nosync: This option is used for internal performance testing and is currently unsupported. Use at your own risk.
- O_DIRECT: 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.

  Prior to MySQL 5.7.25, this setting is not suitable for file systems such as XFS and EXT4, which require an fsync() system call to synchronize file system metadata changes. If you are not sure whether your file system requires an fsync() system call to synchronize file system metadata changes, use O_DIRECT instead.

  As of MySQL 5.7.25, fsync() is called 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.

The `innodb_flush_method` options for Windows systems include:

- `async_unbuffered`: `InnoDB` uses Windows asynchronous I/O and non-buffered I/O. `async_unbuffered` is the default setting on Windows systems.

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

- `normal`: `InnoDB` uses simulated asynchronous I/O and buffered I/O.

- `unbuffered`: `InnoDB` uses simulated asynchronous I/O and non-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 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 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_flush_neighbors`

Command-Line Format	`--innodb-flush-neighbors=#`
System Variable	`innodb_flush_neighbors`
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`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.
- The default 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 turn this setting off to spread out write operations. For related information, see Section 14.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_flush_sync`

Command-Line Format	`--innodb-flush-sync[={OFF|ON}]`
System Variable	`innodb_flush_sync`
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	`ON`

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

For information about configuring the `innodb_flush_sync` variable, see Section 14.8.8, “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
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 14.8.3.5, “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
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
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 14.22.2, “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 because replication settings such as `relay_log_info_repository=TABLE` and `master_info_repository=TABLE` store information in `InnoDB` tables.

- `innodb_ft_aux_table`

System Variable	`innodb_ft_aux_table`
Scope	Global
Dynamic	Yes
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 14.16.4, “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
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
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
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 12.10.4, “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
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 12.10.6, “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
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 12.10.6, “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
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 12.10.6, “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
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
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 12.10.4, “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
Type	Integer
Default Value	2
Minimum Value	1
Maximum Value	32

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 14.6.2.4, “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

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
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 12.10.4, “Full-Text Stopwords”.

- innodb_io_capacity

Command-Line Format	--innodb-io-capacity=#

System Variable	`innodb_io_capacity`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`200`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`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 14.8.8, “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
Type	Integer
Default Value	`see description`
Minimum Value	`100`
Maximum Value (32-bit platforms)	`2**32-1`
Maximum Value (Unix, 64-bit platforms)	`2**64-1`
Maximum Value (Windows, 64-bit platforms)	`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.

For information about configuring the innodb_io_capacity_max variable, see Section 14.8.8, “Configuring InnoDB I/O Capacity”.

- innodb_large_prefix

Command-Line Format	--innodb-large-prefix[={OFF|ON}]
Deprecated	Yes
System Variable	innodb_large_prefix
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	ON

When this option is enabled, index key prefixes longer than 767 bytes (up to 3072 bytes) are allowed for InnoDB tables that use DYNAMIC or COMPRESSED row format. See Section 14.23, “InnoDB Limits” for maximums associated with index key prefixes under various settings.

For tables that use REDUNDANT or COMPACT row format, this option does not affect the permitted index key prefix length.

innodb_large_prefix is enabled by default in MySQL 5.7. This change coincides with the default value change for innodb_file_format, which is set to Barracuda by default in MySQL 5.7. Together, these default value changes allow larger index key prefixes to be created when using DYNAMIC or COMPRESSED row format. If either option is set to a non-default value, index key prefixes larger than 767 bytes are silently truncated.

innodb_large_prefix is deprecated; expect it to be removed in a future release. innodb_large_prefix was introduced to disable large index key prefixes for compatibility with earlier versions of InnoDB that do not support large index key prefixes.

- innodb_limit_optimistic_insert_debug

Command-Line Format	--innodb-limit-optimistic-insert-debug=#
System Variable	innodb_limit_optimistic_insert_debug
Scope	Global
Dynamic	Yes

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
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 14.22.4, “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 only. 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 14.7.5.2, “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 5.1.8.1, “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_locks_unsafe_for_binlog

Command-Line Format	--innodb-locks-unsafe-for-binlog[={OFF|ON}]
Deprecated	Yes
System Variable	innodb_locks_unsafe_for_binlog
Scope	Global
Dynamic	No
Type	Boolean
Default Value	OFF

This variable affects how InnoDB uses gap locking for searches and index scans. innodb_locks_unsafe_for_binlog is deprecated; expect it to be removed in a future MySQL release.

Normally, InnoDB uses an algorithm called next-key locking that combines index-row locking with gap locking. InnoDB performs row-level locking in such a way that when it searches or scans a table index, it sets shared or exclusive locks on the index records it encounters. Thus, row-level locks are actually index-record locks. In addition, a next-key lock on an index record also affects the gap before the index record. That is, a next-key lock is an index-record lock plus a gap lock on the gap preceding the index record. If one session has a shared or exclusive lock on record R in an index, another session cannot insert a new index record in the gap immediately before R in the index order. See Section 14.7.1, “InnoDB Locking”.

By default, the value of innodb_locks_unsafe_for_binlog is 0 (disabled), which means that gap locking is enabled: InnoDB uses next-key locks for searches and index scans. To enable the variable,

set it to 1. This causes gap locking to be disabled: InnoDB uses only index-record locks for searches and index scans.

Enabling innodb_locks_unsafe_for_binlog does not disable the use of gap locking for foreign-key constraint checking or duplicate-key checking.

The effects of enabling innodb_locks_unsafe_for_binlog are the same as setting the transaction isolation level to READ COMMITTED, with these exceptions:

- Enabling innodb_locks_unsafe_for_binlog is a global setting and affects all sessions, whereas the isolation level can be set globally for all sessions, or individually per session.
- innodb_locks_unsafe_for_binlog can be set only at server startup, whereas the isolation level can be set at startup or changed at runtime.

READ COMMITTED therefore offers finer and more flexible control than innodb_locks_unsafe_for_binlog. For more information about the effect of isolation level on gap locking, see Section 14.7.2.1, “Transaction Isolation Levels”.

Enabling innodb_locks_unsafe_for_binlog may cause phantom problems because other sessions can insert new rows into the gaps when gap locking is disabled. Suppose that there is an index on the id column of the child table and that you want to read and lock all rows from the table having an identifier value larger than 100, with the intention of updating some column in the selected rows later:

```
SELECT * FROM child WHERE id > 100 FOR UPDATE;
```

The query scans the index starting from the first record where the id is greater than 100. If the locks set on the index records in that range do not lock out inserts made in the gaps, another session can insert a new row into the table. Consequently, if you were to execute the same SELECT again within the same transaction, you would see a new row in the result set returned by the query. This also means that if new items are added to the database, InnoDB does not guarantee serializability. Therefore, if innodb_locks_unsafe_for_binlog is enabled, InnoDB guarantees at most an isolation level of READ COMMITTED. (Conflict serializability is still guaranteed.) For more information about phantoms, see Section 14.7.4, “Phantom Rows”.

Enabling innodb_locks_unsafe_for_binlog has additional effects:

- For UPDATE or DELETE statements, InnoDB holds locks only for rows that it updates or deletes. Record locks for nonmatching rows are released after MySQL has evaluated the WHERE condition. This greatly reduces the probability of deadlocks, but they can still happen.

- For UPDATE statements, if a row is already locked, InnoDB performs a “semi-consistent” read, returning the latest committed version to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE. If the row matches (must be updated), MySQL reads the row again and this time InnoDB either locks it or waits for a lock on it.

Consider the following example, beginning with this table:

```
CREATE TABLE t (a INT NOT NULL, b INT) ENGINE = InnoDB;
INSERT INTO t VALUES (1,2),(2,3),(3,2),(4,3),(5,2);
COMMIT;
```

In this case, table has no indexes, so searches and index scans use the hidden clustered index for record locking (see Section 14.6.2.1, “Clustered and Secondary Indexes”).

Suppose that one client performs an UPDATE using these statements:

```
SET autocommit = 0;
UPDATE t SET b = 5 WHERE b = 3;
```

Suppose also that a second client performs an UPDATE by executing these statements following those of the first client:

```
SET autocommit = 0;
UPDATE t SET b = 4 WHERE b = 2;
```

As InnoDB executes each UPDATE, it first acquires an exclusive lock for each row, and then determines whether to modify it. If InnoDB does not modify the row and innodb_locks_unsafe_for_binlog is enabled, it releases the lock. Otherwise, InnoDB retains the lock until the end of the transaction. This affects transaction processing as follows.

If innodb_locks_unsafe_for_binlog is disabled, the first UPDATE acquires x-locks and does not release any of them:

```
x-lock(1,2); retain x-lock
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); retain x-lock
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); retain x-lock
```

The second UPDATE blocks as soon as it tries to acquire any locks (because the first update has retained locks on all rows), and does not proceed until the first UPDATE commits or rolls back:

```
x-lock(1,2); block and wait for first UPDATE to commit or roll back
```

If innodb_locks_unsafe_for_binlog is enabled, the first UPDATE acquires x-locks and releases those for rows that it does not modify:

```
x-lock(1,2); unlock(1,2)
x-lock(2,3); update(2,3) to (2,5); retain x-lock
x-lock(3,2); unlock(3,2)
x-lock(4,3); update(4,3) to (4,5); retain x-lock
x-lock(5,2); unlock(5,2)
```

For the second UPDATE, InnoDB does a "semi-consistent" read, returning the latest committed version of each row to MySQL so that MySQL can determine whether the row matches the WHERE condition of the UPDATE:

```
x-lock(1,2); update(1,2) to (1,4); retain x-lock
x-lock(2,3); unlock(2,3)
x-lock(3,2); update(3,2) to (3,4); retain x-lock
x-lock(4,3); unlock(4,3)
x-lock(5,2); update(5,2) to (5,4); retain x-lock
```

- innodb_log_buffer_size

Command-Line Format	--innodb-log-buffer-size=#
System Variable	innodb_log_buffer_size
Scope	Global
Dynamic	No
Type	Integer
Default Value	16777216
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 value changed from 8MB to 16MB with the introduction of 32KB and 64KB `innodb_page_size` values. 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 8.5.4, “Optimizing InnoDB Redo Logging”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_log_checkpoint_now`

Command-Line Format	`--innodb-log-checkpoint-now[={OFF|ON}]`
System Variable	`innodb_log_checkpoint_now`
Scope	Global
Dynamic	Yes
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
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
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 14.9.1.6, “Compression for OLTP Workloads”.

- `innodb_log_file_size`

Command-Line Format	`--innodb-log-file-size=#`
System Variable	`innodb_log_file_size`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`50331648`
Minimum Value (≥ 5.7.11)	`4194304`
Minimum Value (≤ 5.7.10)	`1048576`
Maximum Value	`512GB / innodb_log_files_in_group`
Unit	bytes

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` value was increased from 1MB to 4MB in MySQL 5.7.11.

For related information, see Redo Log File Configuration. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_log_files_in_group`

Command-Line Format	`--innodb-log-files-in-group=#`
System Variable	`innodb_log_files_in_group`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`2`
Minimum Value	`2`
Maximum Value	`100`

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 File Configuration.

- `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
Type	Directory name

The directory path to the InnoDB redo log files, whose number is specified by innodb_log_files_in_group. If you do not specify any InnoDB log variables, the default is to create two files named ib_logfile0 and ib_logfile1 in the MySQL data directory. Log file size is given by the innodb_log_file_size system variable.

For related information, see Redo Log File Configuration.

- innodb_log_write_ahead_size

Command-Line Format	--innodb-log-write-ahead-size=#
System Variable	innodb_log_write_ahead_size
Scope	Global
Dynamic	Yes
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 8.5.4, “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
Type	Integer
Default Value	`1024`
Minimum Value	`100`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`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 14.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “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
Type	Numeric
Default Value	`75`

Minimum Value	`0`
Maximum Value	`99.99`

`InnoDB` tries to flush data from the buffer pool so that the percentage of dirty pages does not exceed this value. The default value is 75.

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 14.8.3.5, “Configuring Buffer Pool Flushing”.

For related information, see Section 14.8.3.5, “Configuring Buffer Pool Flushing”. For general I/O tuning advice, see Section 8.5.8, “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
Type	Numeric
Default Value	`0`
Minimum Value	`0`
Maximum Value	`99.99`

Defines a low water mark representing the percentage of dirty pages at which preflushing is enabled to control the dirty page ratio. The default 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 14.8.3.5, “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
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 14.8.10, “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
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	10000000
Unit	milliseconds

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 14.8.10, “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
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
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 14.8.12, “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
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 14.16.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

`innodb_monitor_disable='latch'` disables statistics collection for `SHOW ENGINE INNODB MUTEX`. For more information, see Section 13.7.5.15, “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
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 14.16.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

`innodb_monitor_enable='latch'` enables statistics collection for `SHOW ENGINE INNODB MUTEX`. For more information, see Section 13.7.5.15, “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
Type	Enumeration
Default Value	`NULL`

Valid Values	`counter`
	`module`
	`pattern`
	`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 14.16.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

`innodb_monitor_reset='latch'` resets statistics reported by `SHOW ENGINE INNODB MUTEX`. For more information, see Section 13.7.5.15, “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
Type	Enumeration
Default Value	`NULL`

Valid Values	`counter`
	`module`
	`pattern`
	`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 14.16.6, “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
Type	Boolean
Default Value	`OFF`

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.

As of MySQL 5.7.17, **CMake** sets the default `WITH_NUMA` value based on whether the current platform has `NUMA` support. For more information, see Section 2.9.7, “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
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 14.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 14.5.1, “Buffer Pool”.

- `innodb_old_blocks_time`

Command-Line Format	`--innodb-old-blocks-time=#`
System Variable	`innodb_old_blocks_time`
Scope	Global
Dynamic	Yes
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 14.8.3.3, “Making the Buffer Pool Scan Resistant”. For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 14.5.1, “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
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	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.

- innodb_optimize_fulltext_only

Command-Line Format	--innodb-optimize-fulltext-only[={OFF|ON}]
System Variable	innodb_optimize_fulltext_only
Scope	Global
Dynamic	Yes
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
Type	Integer
Default Value	4
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. A single page cleaner thread was introduced in MySQL 5.6 to offload buffer pool flushing work from the InnoDB master thread. In MySQL 5.7, InnoDB provides support for multiple page cleaner threads. A value of 1 maintains the pre-MySQL 5.7

configuration in which there is a single page cleaner thread. 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 was changed from 1 to 4 in MySQL 5.7. If the number of page cleaner threads exceeds the number of buffer pool instances, `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 is extended to shutdown and recovery phases in MySQL 5.7.

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
Type	Enumeration
Default Value	`16384`

Valid Values	`4096`
	`8192`
	`16384`
	`32768`
	`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 14.8.1, “InnoDB Startup Configuration”.

Support for 32KB and 64KB page sizes was added in MySQL 5.7. 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=32k`, extent size is 2MB. For

innodb_page_size=64KB, extent size is 4MB. innodb_log_buffer_size should be set to at least 16M (the default) 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 8.5.8, “Optimizing InnoDB Disk I/O”.

- innodb_print_all_deadlocks

Command-Line Format	--innodb-print-all-deadlocks[={OFF|ON}]
System Variable	innodb_print_all_deadlocks
Scope	Global
Dynamic	Yes
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 command. 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 14.7.5, “Deadlocks in InnoDB”.

- innodb_purge_batch_size

Command-Line Format	`--innodb-purge-batch-size=#`
System Variable	`innodb_purge_batch_size`
Scope	Global
Dynamic	Yes
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 14.8.10, “Purge Configuration”.

- `innodb_purge_threads`

Command-Line Format	`--innodb-purge-threads=#`
System Variable	`innodb_purge_threads`
Scope	Global
Dynamic	No
Type	Integer
Default Value	`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 14.8.10, “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
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
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 14.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “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
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 14.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)”. For general I/O tuning advice, see Section 8.5.8, “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
Type	Integer

Default Value	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 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 8.5.8, “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.

- innodb_read_only

Command-Line Format	--innodb-read-only[={OFF|ON}]
System Variable	innodb_read_only
Scope	Global
Dynamic	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 14.8.2, “Configuring InnoDB for Read-Only Operation”.

- innodb_replication_delay

Command-Line Format	--innodb-replication-delay=#
System Variable	innodb_replication_delay
Scope	Global
Dynamic	Yes
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
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 14.22.4, “InnoDB Error Handling”.

- `innodb_rollback_segments`

Command-Line Format	`--innodb-rollback-segments=#`
System Variable	`innodb_rollback_segments`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`128`
Minimum Value	`1`
Maximum Value	`128`

Defines the number of rollback segments used by `InnoDB` 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 14.6.7, “Undo Logs”.

One rollback segment is always assigned to the system tablespace, and 32 rollback segments are reserved for use by temporary tables and reside in the temporary tablespace (`ibtmp1`). To allocate additional rollback segment, `innodb_rollback_segments` must be set to a value greater than 33. If you configure separate undo tablespaces, the rollback segment in the system tablespace is rendered inactive.

When `innodb_rollback_segments` is set to 32 or less, `InnoDB` assigns one rollback segment to the system tablespace and 32 to the temporary tablespace.

When `innodb_rollback_segments` is set to a value greater than 32, `InnoDB` assigns one rollback segment to the system tablespace, 32 to the temporary tablespace, and additional rollback segments to undo tablespaces, if present. If undo tablespaces are not present, additional rollback segments are assigned to the system tablespace.

Although you can increase or decrease the number of rollback segments used by `InnoDB`, the number of rollback segments physically present in the system never decreases. Thus, you might start with a low value and gradually increase it to avoid allocating rollback segments that are not required. The `innodb_rollback_segments` default and maximum value is 128.

For related information, see Section 14.3, “InnoDB Multi-Versioning”. For information about configuring separate undo tablespaces, see Section 14.6.3.4, “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
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2**23-1`

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_sort_buffer_size`

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

This variable defines:

- The sort buffer size for online DDL operations that create or rebuild secondary indexes.
- 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 related information, see Section 14.13.3, “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
Type	Integer
Default Value	`6`
Minimum Value	`0`
Maximum Value (64-bit platforms)	`2**64-1`
Maximum Value (32-bit platforms)	`2**32-1`

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. For more information, see Section 14.8.9, “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
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_PERSISTENT=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 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- innodb_stats_include_delete_marked

Command-Line Format	--innodb-stats-include-delete-marked[={OFF|ON}]
Introduced	5.7.17
System Variable	innodb_stats_include_delete_marked
Scope	Global
Dynamic	Yes
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 14.8.11.1, “Configuring Persistent Optimizer Statistics Parameters”.

- innodb_stats_method

Command-Line Format	--innodb-stats-method=value
System Variable	innodb_stats_method
Scope	Global
Dynamic	Yes
Type	Enumeration
Default Value	nulls_equal
Valid Values	nulls_equal nulls_unequal 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 8.3.7, “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
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 14.8.11.2, “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 INFORMATION_SCHEMA.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 5.1.8.1, “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
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 14.8.11.1, “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	

Type	Yes 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 14.8.11.1, “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 14.8.11.3, “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_sample_pages`

Command-Line Format	`--innodb-stats-sample-pages=#`
Deprecated	Yes
System Variable	`innodb_stats_sample_pages`
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	`8`
Minimum Value	`1`
Maximum Value	`2**64-1`

Deprecated. Use `innodb_stats_transient_sample_pages` 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
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 14.8.11.2, “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 14.8.11.3, “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`. For more information, see Section 14.8.11.2, “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
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 14.18.2, “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
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 14.18.2, “Enabling InnoDB Monitors”.

- innodb_strict_mode

Command-Line Format	--innodb-strict-mode[={OFF|ON}]
System Variable	innodb_strict_mode
Scope	Global, Session
Dynamic	Yes
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 5.1.8.1, “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.

- innodb_support_xa

Command-Line Format	--innodb-support-xa[={OFF|ON}]
Deprecated	5.7.10
System Variable	innodb_support_xa
Scope	Global, Session
Dynamic	Yes
Type	Boolean
Default Value	ON

Enables InnoDB support for two-phase commit in XA transactions, causing an extra disk flush for transaction preparation. The XA mechanism is used internally and is essential for any server that has its binary log turned on and is accepting changes to its data from more than one thread. If you disable innodb_support_xa, transactions can be written to the binary log in a different order than the live database is committing them, which can produce different data when the binary log is replayed in disaster recovery or on a replica. Do not disable innodb_support_xa on a replication source server unless you have an unusual setup where only one thread is able to change data.

innodb_support_xa is deprecated; expect it to be removed in a future MySQL release. InnoDB support for two-phase commit in XA transactions is always enabled as of MySQL 5.7.10. Disabling innodb_support_xa is no longer permitted as it makes replication unsafe and prevents performance gains associated with binary log group commit.

- innodb_sync_array_size

Command-Line Format	--innodb-sync-array-size=#
System Variable	innodb_sync_array_size
Scope	Global
Dynamic	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
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
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.

Previously, enabling InnoDB sync debug checking required that the Debug Sync facility be enabled using the ENABLE_DEBUG_SYNC **CMake** option. This requirement was removed in MySQL 5.7 with the

introduction of this variable.

- `innodb_table_locks`

Command-Line Format	`--innodb-table-locks[={OFF|ON}]`
System Variable	`innodb_table_locks`
Scope	Global, Session
Dynamic	Yes
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 14.7, “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
Type	String
Default Value	`ibtmp1:12M:autoextend`

Defines the relative path, name, size, and attributes of `InnoDB` temporary tablespace data files. If you do not specify a value for `innodb_temp_data_file_path`, the default behavior is to create a single, auto-extending data file named `ibtmp1` in the MySQL data directory. The initial file size is slightly larger than 12MB.

The full syntax for a 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 temporary tablespace data file cannot have the same name as another `InnoDB` data file. Any inability or error creating a temporary tablespace data file is treated as fatal and server startup is refused. The temporary tablespace has a dynamically generated space ID, which can change on each server restart.

File sizes are specified KB, MB or GB (1024MB) by appending `K`, `M` or `G` to the size value. The sum of the sizes of the files must be slightly larger than 12MB.

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

The `autoextend` and `max` attributes can be used only for the data file that is 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
```

If you specify the `autoextend` option, `InnoDB` extends the data file if it runs out of free space. The `autoextend` increment is 64MB by default. To modify the increment, change the `innodb_autoextend_increment` system variable.

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

The temporary tablespace is shared by all non-compressed `InnoDB` temporary tables. Compressed temporary tables reside in file-per-table tablespace files created in the temporary file directory, which is defined by the `tmpdir` configuration option.

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
```

Metadata about active `InnoDB` temporary tables is located in `INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO`.

For related information, see Section 14.6.3.5, “The Temporary Tablespace”.

- innodb_thread_concurrency

Command-Line Format	`--innodb-thread-concurrency=#`
System Variable	`innodb_thread_concurrency`
Scope	Global
Dynamic	Yes
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. Once the limit is reached, additional threads are placed into a "First In, First Out" (FIFO) queue for waiting threads. 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 14.8.5, “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
Type	Integer
Default Value	`10000`
Minimum Value	`0`

Maximum Value	1000000
Unit	microseconds

Defines 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 14.8.5, “Configuring Thread Concurrency for InnoDB”.

- innodb_tmpdir

Command-Line Format	--innodb-tmpdir=dir_name
Introduced	5.7.11
System Variable	innodb_tmpdir
Scope	Global, Session
Dynamic	Yes
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 14.13.3, “Online DDL Space Requirements”. For information about online `ALTER TABLE` operations, see Section 14.13, “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
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
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
Type	Directory name

The path where InnoDB creates undo tablespaces. Typically used to place undo logs on a different storage device. Used in conjunction with innodb_rollback_segments and innodb_undo_tablespaces.

There is no default value (it is NULL). If a path is not specified, undo tablespaces are created in the MySQL data directory, as defined by datadir.

For more information, see Section 14.6.3.4, “Undo Tablespaces”.

- innodb_undo_log_truncate

Command-Line Format	--innodb-undo-log-truncate[={OFF|ON}]
System Variable	innodb_undo_log_truncate
Scope	Global
Dynamic	Yes
Type	Boolean
Default Value	OFF

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 and two redo-enabled undo logs configured to use undo tablespaces. This means that innodb_undo_tablespaces must be set to a value equal to or greater than 2, and innodb_rollback_segments must set to a value equal to or greater than 35.

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_logs

Command-Line Format	--innodb-undo-logs=#
Deprecated	5.7.19
System Variable	innodb_undo_logs
Scope	Global
Dynamic	Yes
Type	Integer
Default Value	128
Minimum Value	1
Maximum Value	128

> **Note**
>
> innodb_undo_logs is deprecated; expect it to be removed in a future MySQL release.

Defines the number of rollback segments used by InnoDB. The innodb_undo_logs option is an alias for innodb_rollback_segments. For more information, see the description of innodb_rollback_segments.

- innodb_undo_tablespaces

Command-Line Format	--innodb-undo-tablespaces=#
Deprecated	5.7.21
System Variable	innodb_undo_tablespaces
Scope	Global
Dynamic	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	95

The number of undo tablespaces used by InnoDB. The default value is 0.

> **Note**
>
> innodb_undo_tablespaces is deprecated; expect it to be removed in a future MySQL release.

Because undo logs can become large during long-running transactions, having undo logs in multiple tablespaces reduces the maximum size of any one tablespace. The undo tablespace files are created in the location defined by innodb_undo_directory, with names in the form of undo***N***, where ***N*** is a sequential series of integers (including leading zeros) representing the space ID.

The initial size of an undo tablespace file depends on the innodb_page_size value. For the default 16KB InnoDB page size, the initial undo tablespace file size is 10MiB. For 4KB, 8KB, 32KB, and 64KB page sizes, the initial undo tablespace files sizes are 7MiB, 8MiB, 20MiB, and 40MiB, respectively.

A minimum of two undo tablespaces is required to enable truncation of undo logs. See Truncating Undo Tablespaces.

> **Important**
>
> innodb_undo_tablespaces 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 setting of 0. Attempting to restart InnoDB with a greater number of undo tablespaces than specified when the MySQL instance was initialized results in a startup failure and an error stating that InnoDB did not find the expected number of undo tablespaces.

32 of 128 rollback segments are reserved for temporary tables, as described in Section 14.6.7, “Undo Logs”. One rollback segment is always assigned to the system tablespace, which leaves 95 rollback segments available for undo tablespaces. This means the innodb_undo_tablespaces maximum limit is 95.

For more information, see Section 14.6.3.4, “Undo Tablespaces”.

- innodb_use_native_aio

Command-Line Format	--innodb-use-native-aio[={OFF|ON}]
System Variable	innodb_use_native_aio
Scope	Global
Dynamic	No

Type	Boolean
Default Value	`ON`

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

The asynchronous I/O capability that `InnoDB` has on Windows systems is available on Linux systems. (Other Unix-like systems continue to use synchronous I/O calls.) This feature improves the scalability of heavily I/O-bound systems, which typically show many pending reads/writes in `SHOW ENGINE INNODB STATUS\G` 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 14.8.7, “Using Asynchronous I/O on Linux”.

- `innodb_version`

  The `InnoDB` version number. In MySQL 5.7, separate version numbering for `InnoDB` does not apply and this value is the same the `version` number of the server.

- `innodb_write_io_threads`

Command-Line Format	`--innodb-write-io-threads=#`
System Variable	`innodb_write_io_threads`
Scope	Global
Dynamic	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 14.8.6, “Configuring the Number of Background InnoDB I/O Threads”. For general I/O tuning advice, see Section 8.5.8, “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 8.5.8, “Optimizing InnoDB Disk I/O”.