

For partitions or subpartitions, you can use the same query with a modified WHERE clause to retrieve index sizes. For example, the following query retrieves index sizes for partitions of table t1:

mysql> **SELECT** **SUM(stat\_value)** **pages,** **index\_name,**

**SUM(stat\_value)\*@@innodb\_page\_size** **size**

**FROM** **mysql.innodb\_index\_stats** **WHERE** **table\_name** **like** **'t1#P%'**

**AND** **stat\_name** **=** **'size'** **GROUP** **BY** **index\_name;**

**15.8.10.2** **Configuring** **Non-Persistent** **Optimizer** **Statistics** **Parameters**

This section describes how to configure non-persistent optimizer statistics. Optimizer statistics are not persisted to disk when [innodb\_stats\_persistent=OFF](#_bookmark2) or when individual tables are created or altered with STATS\_PERSISTENT=0. Instead, statistics are stored in memory, and are lost when the server is shut down. Statistics are also updated periodically by certain operations and under certain conditions.

Optimizer statistics are persisted to disk by default, enabled by the [innodb\_stats\_persistent](#_bookmark2) configuration option. For information about persistent optimizer statistics, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters” .

**Optimizer** **Statistics** **Updates**

Non-persistent optimizer statistics are updated when:

• Running ANALYZE TABLE.

• Running SHOW TABLE STATUS, SHOW INDEX, or querying the Information Schema TABLES or STATISTICS tables with the [innodb\_stats\_on\_metadata](#_bookmark3) option enabled.

The default setting for [innodb\_stats\_on\_metadata](#_bookmark3) is OFF. Enabling [innodb\_stats\_on\_metadata](#_bookmark3) may reduce access speed for schemas that have a large number of tables or indexes, and reduce stability of execution plans for queries that involve InnoDB tables. [innodb\_stats\_on\_metadata](#_bookmark3) is configured globally using a SET statement.

SET GLOBAL innodb\_stats\_on\_metadata=ON

**Note**

[innodb\_stats\_on\_metadata](#_bookmark3) only applies when optimizer statistics are configured to be non-persistent (when [innodb\_stats\_persistent](#_bookmark2) is disabled).

• Starting a mysql client with the --auto-rehash option enabled, which is the default. The auto- rehash option causes all InnoDB tables to be opened, and the open table operations cause statistics to be recalculated.

To improve the start up time of the mysql client and to updating statistics, you can turn off auto- rehash using the --disable-auto-rehash option. The auto-rehash feature enables automatic name completion of database, table, and column names for interactive users.

• A table is first opened.

• InnoDB detects that 1 / 16 of table has been modified since the last time statistics were updated.

**Configuring** **the** **Number** **of** **Sampled** **Pages**

The MySQL query optimizer uses estimated statistics about key distributions to choose the indexes for an execution plan, based on the relative selectivity of the index. When InnoDB updates optimizer statistics, it samples random pages from each index on a table to estimate the cardinality of the index. (This technique is known as random dives.)

To give you control over the quality of the statistics estimate (and thus better information for the query optimizer), you can change the number of sampled pages using the parameter



[innodb\_stats\_transient\_sample\_pages](#_bookmark4). The default number of sampled pages is 8, which could be insufficient to produce an accurate estimate, leading to poor index choices by the query optimizer. This technique is especially important for large tables and tables used in joins. Unnecessary full table scans for such tables can be a substantial performance issue. See Section 8.2.1.23, “Avoiding Full Table Scans” for tips on tuning such queries. [innodb\_stats\_transient\_sample\_pages](#_bookmark4) is a global parameter that can be set at runtime.

The value of [innodb\_stats\_transient\_sample\_pages](#_bookmark4) affects the index sampling for all InnoDB tables and indexes when [innodb\_stats\_persistent=0](#_bookmark2). Be aware of the following potentially significant impacts when you change the index sample size:

• Small values like 1 or 2 can result in inaccurate estimates of cardinality.

• Increasing the [innodb\_stats\_transient\_sample\_pages](#_bookmark4) value might require more disk reads. Values much larger than 8 (say, 100), can cause a significant slowdown in the time it takes to open a table or execute SHOW TABLE STATUS.

• The optimizer might choose very different query plans based on different estimates of index selectivity.

Whatever value of [innodb\_stats\_transient\_sample\_pages](#_bookmark4) works best for a system, set the option and leave it at that value. Choose a value that results in reasonably accurate estimates for all tables in your database without requiring excessive I/O. Because the statistics are automatically recalculated at various times other than on execution of ANALYZE TABLE, it does not make sense to increase the index sample size, run ANALYZE TABLE, then decrease sample size again.

Smaller tables generally require fewer index samples than larger tables. If your database has many large tables, consider using a higher value for [innodb\_stats\_transient\_sample\_pages](#_bookmark4) than if you have mostly smaller tables.

**15.8.10.3** **Estimating** **ANALYZE** **TABLE** **Complexity** **for** **InnoDB** **Tables**

ANALYZE TABLE complexity for InnoDB tables is dependent on:

• The number of pages sampled, as defined by [innodb\_stats\_persistent\_sample\_pages](#_bookmark6).

• The number of indexed columns in a table

• The number of partitions. If a table has no partitions, the number of partitions is considered to be 1. Using these parameters, an approximate formula for estimating ANALYZE TABLE complexity would be:

The value of [innodb\_stats\_persistent\_sample\_pages](#_bookmark6) \* number of indexed columns in a table \* the number of partitions

Typically, the greater the resulting value, the greater the execution time for ANALYZE TABLE.

**Note**

[innodb\_stats\_persistent\_sample\_pages](#_bookmark6) defines the number of pages sampled at a global level. To set the number of pages sampled for an individual table, use the STATS\_SAMPLE\_PAGES option with CREATE TABLE or ALTER TABLE. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters” .

If [innodb\_stats\_persistent=OFF](#_bookmark2), the number of pages sampled is defined by [innodb\_stats\_transient\_sample\_pages](#_bookmark4). See [Section 15.8.10.2,](#_bookmark1) [“Configuring Non-Persistent Optimizer Statistics Parameters”](#_bookmark1) for additional information.

For a more in-depth approach to estimating ANALYZE TABLE complexity, consider the following example.



In [Big O notation](http://en.wikipedia.org/wiki/Big_O_notation), ANALYZE TABLE complexity is described as:

O(n\_sample

\* (n\_cols\_in\_uniq\_i

+ n\_cols\_in\_non\_uniq\_i

+ n\_cols\_in\_pk \* (1 + n\_non\_uniq\_i))

\* n\_part)

where:

• n\_sample is the number of pages sampled (defined by

[innodb\_stats\_persistent\_sample\_pages](#_bookmark6))

• n\_cols\_in\_uniq\_i is total number of all columns in all unique indexes (not counting the primary key columns)

• n\_cols\_in\_non\_uniq\_i is the total number of all columns in all nonunique indexes

• n\_cols\_in\_pk is the number of columns in the primary key (if a primary key is not defined, InnoDB creates a single column primary key internally)

• n\_non\_uniq\_i is the number of nonunique indexes in the table

• n\_part is the number of partitions. If no partitions are defined, the table is considered to be a single partition.

Now, consider the following table (table t), which has a primary key (2 columns), a unique index (2 columns), and two nonunique indexes (two columns each):

CREATE TABLE t (

a INT,

b INT,

c INT,

d INT,

e INT,

f INT,

g INT,

h INT,

PRIMARY KEY (a, b),

UNIQUE KEY i1uniq (c, d),

KEY i2nonuniq (e, f),

KEY i3nonuniq (g, h)

);

For the column and index data required by the algorithm described above, query the

mysql.innodb\_index\_stats persistent index statistics table for table t. The n\_diff\_pfx% statistics show the columns that are counted for each index. For example, columns a and b are counted for the primary key index. For the nonunique indexes, the primary key columns (a,b) are counted in addition to the user defined columns.

**Note**

For additional information about the InnoDB persistent statistics tables, see

Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”

mysql> **SELECT** **index\_name,** **stat\_name,** **stat\_description**

**FROM** **mysql.innodb\_index\_stats** **WHERE**

**database\_name='test'** **AND**

**table\_name='t'** **AND**

**stat\_name** **like** **'n\_diff\_pfx%';**

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

| index\_name | stat\_name | stat\_description |

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

|  |  |  |
| --- | --- | --- |
| | PRIMARY  | PRIMARY  | i1uniq  | i1uniq | | n\_diff\_pfx01 | a  | n\_diff\_pfx02 | a,b  | n\_diff\_pfx01 | c  | n\_diff\_pfx02 | c,d | |  |  |  | |



n\_diff\_pfx01 |

n\_diff\_pfx02 |

n\_diff\_pfx03 |

n\_diff\_pfx04 |

n\_diff\_pfx01 |

n\_diff\_pfx02 |

n\_diff\_pfx03 |

n\_diff\_pfx04 |

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

Based on the index statistics data shown above and the table definition, the following values can be determined:

i2nonuniq

i2nonuniq

i2nonuniq

i2nonuniq

i3nonuniq

i3nonuniq

i3nonuniq

i3nonuniq

e

e,f

e,f,a

e,f,a,b

g

g,h

g,h,a

g,h,a,b

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

• n\_cols\_in\_uniq\_i, the total number of all columns in all unique indexes not counting the primary key columns, is 2 (c and d)

• n\_cols\_in\_non\_uniq\_i, the total number of all columns in all nonunique indexes, is 4 (e, f, g and h)

• n\_cols\_in\_pk, the number of columns in the primary key, is 2 (a and b)

• n\_non\_uniq\_i, the number of nonunique indexes in the table, is 2 (i2nonuniq and i3nonuniq))

• n\_part, the number of partitions, is 1.

You can now calculate innodb\_stats\_persistent\_sample\_pages \* (2 +

4 + 2 \* (1 + 2)) \* 1 to determine the number of leaf pages that are scanned. With innodb\_stats\_persistent\_sample\_pages set to the default value of 20, and with a default page size of 16 KiB ([innodb\_page\_size](#_bookmark7)=16384), you can then estimate that 20 \* 12 \* 16384 bytes are read for table t, or about 4 MiB.

**Note**

All 4 MiB may not be read from disk, as some leaf pages may already be cached in the buffer pool.

**15.8.11** **Configuring** **the** **Merge** **Threshold** **for** **Index** **Pages**

You can configure the MERGE\_THRESHOLD value for index pages. If the “page-full” percentage for an index page falls below the MERGE\_THRESHOLD value when a row is deleted or when a row is shortened by an UPDATE operation, InnoDB attempts to merge the index page with a neighboring index page. The default MERGE\_THRESHOLD value is 50, which is the previously hardcoded value. The minimum MERGE\_THRESHOLD value is 1 and the maximum value is 50.

When the “page-full” percentage for an index page falls below 50%, which is the default MERGE\_THRESHOLD setting, InnoDB attempts to merge the index page with a neighboring page. If both pages are close to 50% full, a page split can occur soon after the pages are merged. If this merge- split behavior occurs frequently, it can have an adverse affect on performance. To avoid frequent merge-splits, you can lower the MERGE\_THRESHOLD value so that InnoDB attempts page merges at a lower “page-full” percentage. Merging pages at a lower page-full percentage leaves more room in index pages and helps reduce merge-split behavior.

The MERGE\_THRESHOLD for index pages can be defined for a table or for individual indexes. A MERGE\_THRESHOLD value defined for an individual index takes priority over a MERGE\_THRESHOLD value defined for the table. If undefined, the MERGE\_THRESHOLD value defaults to 50.

**Setting** **MERGE\_THRESHOLD** **for** **a** **Table**

You can set the MERGE\_THRESHOLD value for a table using the *table\_option* COMMENT clause of the CREATE TABLE statement. For example:

CREATE TABLE t1 (

id INT,



KEY id\_index (id)

) COMMENT='MERGE\_THRESHOLD=45';

You can also set the MERGE\_THRESHOLD value for an existing table using the *table\_option* COMMENT clause with ALTER TABLE:

CREATE TABLE t1 (

id INT,

KEY id\_index (id)

);

ALTER TABLE t1 COMMENT='MERGE\_THRESHOLD=40';

**Setting** **MERGE\_THRESHOLD** **for** **Individual** **Indexes**

To set the MERGE\_THRESHOLD value for an individual index, you can use the *index\_option* COMMENT clause with CREATE TABLE, ALTER TABLE, or CREATE INDEX, as shown in the following examples:

• Setting MERGE\_THRESHOLD for an individual index using CREATE TABLE:

CREATE TABLE t1 (

id INT,

KEY id\_index (id) COMMENT 'MERGE\_THRESHOLD=40'

);

• Setting MERGE\_THRESHOLD for an individual index using ALTER TABLE:

CREATE TABLE t1 (

id INT,

KEY id\_index (id)

);

ALTER TABLE t1 DROP KEY id\_index;

ALTER TABLE t1 ADD KEY id\_index (id) COMMENT 'MERGE\_THRESHOLD=40';

• Setting MERGE\_THRESHOLD for an individual index using CREATE INDEX:

CREATE TABLE t1 (id INT);

CREATE INDEX id\_index ON t1 (id) COMMENT 'MERGE\_THRESHOLD=40';

**Note**

You cannot modify the MERGE\_THRESHOLD value at the index level for GEN\_CLUST\_INDEX, which is the clustered index created by InnoDB when an InnoDB table is created without a primary key or unique key index. You can only modify the MERGE\_THRESHOLD value for GEN\_CLUST\_INDEX by setting MERGE\_THRESHOLD for the table.

**Querying** **the** **MERGE\_THRESHOLD** **Value** **for** **an** **Index**

The current MERGE\_THRESHOLD value for an index can be obtained by querying the INNODB\_INDEXES table. For example:

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_INDEXES** **WHERE** **NAME='id\_index'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INDEX\_ID: 91

NAME: id\_index

TABLE\_ID: 68

TYPE: 0

N\_FIELDS: 1

PAGE\_NO: 4

SPACE: 57

MERGE\_THRESHOLD: 40

You can use SHOW CREATE TABLE to view the MERGE\_THRESHOLD value for a table, if explicitly defined using the *table\_option* COMMENT clause:



**WHERE** **NAME** **like** **'%index\_page\_merge%';**

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

| NAME | COMMENT |

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

| index\_page\_merge\_attempts | Number of index page merge attempts |

| index\_page\_merge\_successful | Number of successful index page merges |

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

mysql> **SHOW** **CREATE** **TABLE** **t2** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: t2

Create Table: CREATE TABLE `t2` (

`id` int(11) DEFAULT NULL,

KEY `id\_index` (`id`) COMMENT 'MERGE\_THRESHOLD=40'

) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4

**Note**

A MERGE\_THRESHOLD value defined at the index level takes priority

over a MERGE\_THRESHOLD value defined for the table. If undefined, MERGE\_THRESHOLD defaults to 50% (MERGE\_THRESHOLD=50, which is the previously hardcoded value.

Likewise, you can use SHOW INDEX to view the MERGE\_THRESHOLD value for an index, if explicitly defined using the *index\_option*

COMMENT clause:

mysql> **SHOW** **INDEX** **FROM** **t2** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Table: t2

Non\_unique: 1

Key\_name: id\_index

Seq\_in\_index: 1

Column\_name: id

Collation: A

Cardinality: 0

Sub\_part: NULL

Packed: NULL

Null: YES

Index\_type: BTREE

Comment:

Index\_comment: MERGE\_THRESHOLD=40

**Measuring** **the** **Effect** **of** **MERGE\_THRESHOLD** **Settings**

The INNODB\_METRICS table provides two counters that can be used to measure the effect of a MERGE\_THRESHOLD setting on index page merges.

mysql> **SELECT** **NAME,** **COMMENT** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS**

When lowering the MERGE\_THRESHOLD value, the objectives are:

• A smaller number of page merge attempts and successful page merges

• A similar number of page merge attempts and successful page merges

A MERGE\_THRESHOLD setting that is too small could result in large data files due to an excessive amount of empty page space.

For information about using INNODB\_METRICS counters, see Section 15.15.6, “InnoDB INFORMATION\_SCHEMA Metrics Table” .

**15.8.12** **Enabling** **Automatic** **Configuration** **for** **a** **Dedicated** **MySQL** **Server**

When innodb\_dedicated\_server is enabled, InnoDB automatically configures the following variables:

• innodb\_buffer\_pool\_size



• innodb\_redo\_log\_capacity or, prior to MySQL 8.0.30, [innodb\_log\_file\_size](#_bookmark10) and [innodb\_log\_files\_in\_group](#_bookmark11).

**Note**

innodb\_log\_file\_size and innodb\_log\_files\_in\_group are deprecated in MySQL 8.0.30. These variables are superseded by the innodb\_redo\_log\_capacity variable.

• [innodb\_flush\_method](#_bookmark12)

Only consider enabling [innodb\_dedicated\_server](#_bookmark13) if the MySQL instance resides on a

dedicated server where it can use all available system resources. For example, consider enabling [innodb\_dedicated\_server](#_bookmark13) if you run MySQL Server in a Docker container or dedicated VM that only runs MySQL. Enabling [innodb\_dedicated\_server](#_bookmark13) is not recommended if the MySQL instance shares system resources with other applications.

The information that follows describes how each variable is automatically configured.

• [innodb\_buffer\_pool\_size](#_bookmark14)

Buffer pool size is configured according to the amount of memory detected on the server.

**Table** **15.8** **Automatically** **Configured** **Buffer** **Pool** **Size**

|  |  |
| --- | --- |
| **Detected** **Server** **Memory** | **Buffer** **Pool** **Size** |
| Less than 1GB | 128MB (the default value) |
| 1GB to 4GB | *detected* *server* *memory* \* 0.5 |
| Greater than 4GB | *detected* *server* *memory* \* 0.75 |

[innodb\_redo\_log\_capacity](#_bookmark15)

•

Redo log capacity is configured according to the amount of memory detected on the server and, in some cases, whether [innodb\_buffer\_pool\_size](#_bookmark14) is configured explicitly. If [innodb\_buffer\_pool\_size](#_bookmark14) is not configured explicitly, the default value is assumed.

**Warning**

Automatic redo log capacity configuration behavior is undefined if [innodb\_buffer\_pool\_size](#_bookmark14) is set to a value larger than the detected amount of server memory.

**Table** **15.9** **Automatically** **Configured** **Log** **File** **Size**

|  |  |  |
| --- | --- | --- |
| **Detected** **Server** **Memory** | **Buffer** **Pool** **Size** | **Redo** **Log** **Capacity** |
| Less than 1GB | Not configured | 100MB |
| Less than 1GB | Less than 1GB | 100MB |
| 1GB to 2GB | Not applicable | 100MB |
| 2GB to 4GB | Not configured | 1GB |
| 2GB to 4GB | Any configured value | round(0.5 \* *detected* *server* *memory* in GB) \* 0.5 GB |
| 4GB to 10.66GB | Not applicable | round(0.75 \* *detected* *server* *memory* in GB) \* 0.5  GB |
| 10.66GB to 170.66GB | Not applicable | round(0.5625 \* *detected* *server* *memory* in GB) \* 0.5  GB |



|  |  |  |
| --- | --- | --- |
| **Detected** **Server** **Memory** | **Buffer** **Pool** **Size** | **Redo** **Log** **Capacity** |
| Greater than 170.66GB | Not applicable | 128GB |

• [innodb\_log\_file\_size](#_bookmark10) (deprecated in MySQL 8.0.30)

Log file size is configured according to the automatically configured buffer pool size.

**Table** **15.10** **Automatically** **Configured** **Log** **File** **Size**

|  |  |
| --- | --- |
| **Buffer** **Pool** **Size** | **Log** **File** **Size** |
| Less than 8GB | 512MB |
| 8GB to 128GB | 1024MB |
| Greater than 128GB | 2048MB |

• [innodb\_log\_files\_in\_group](#_bookmark11) (deprecated in MySQL 8.0.30)

The number of log files is configured according to the automatically configured buffer pool size. Automatic configuration of the [innodb\_log\_files\_in\_group](#_bookmark11) variable was added in MySQL

8.0.14.

**Table** **15.11** **Automatically** **Configured** **Number** **of** **Log** **Files**

|  |  |
| --- | --- |
| **Buffer** **Pool** **Size** | **Number** **of** **Log** **Files** |
| Less than 8GB | round(*buffer* *pool* *size*) |
| 8GB to 128GB | round(*buffer* *pool* *size* \* 0.75) |
| Greater than 128GB | 64 |

**Note**

The minimum [innodb\_log\_files\_in\_group](#_bookmark11) value of 2 is enforced if the rounded buffer pool size value is less than 2GB.

• [innodb\_flush\_method](#_bookmark12)

The flush method is set to O\_DIRECT\_NO\_FSYNC when [innodb\_dedicated\_server](#_bookmark13) is enabled. If the O\_DIRECT\_NO\_FSYNC setting is not available, the default [innodb\_flush\_method](#_bookmark12) setting is used.

InnoDB uses O\_DIRECT during flushing I/O, but skips the fsync() system call after each write operation.

**Warning**

Prior to MySQL 8.0.14, 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.

As of MySQL 8.0.14, 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.

If an automatically configured option is configured explicitly in an option file or elsewhere, the explicitly specified setting is used, and a startup warning similar to this is printed to stderr:

[Warning] [000000] InnoDB: Option innodb\_dedicated\_server is ignored for innodb\_buffer\_pool\_size because innodb\_buffer\_pool\_size=134217728 is specified explicitly.

Explicit configuration of one option does not prevent the automatic configuration of other options.

If [innodb\_dedicated\_server](#_bookmark13) is enabled and [innodb\_buffer\_pool\_size](#_bookmark14) is configured explicitly, variables configured based on buffer pool size use the buffer pool size value calculated according to the amount of memory detected on the server rather than the explicitly defined buffer pool size value.

Automatically configured settings are evaluated and reconfigured if necessary each time the MySQL server is started.

**15.9** **InnoDB** **Table** **and** **Page** **Compression**

This section provides information about the InnoDB table compression and InnoDB page compression features. The page compression feature is also referred to as transparent page compression.

Using the compression features of InnoDB, you can create tables where the data is stored in compressed form. Compression can help to improve both raw performance and scalability. The compression means less data is transferred between disk and memory, and takes up less space on disk and in memory. The benefits are amplified for tables with secondary indexes, because index data is compressed also. Compression can be especially important for SSD storage devices, because they tend to have lower capacity than HDD devices.

**15.9.1** **InnoDB** **Table** **Compression**

This section describes InnoDB table compression, which is supported with InnoDB tables that reside in file\_per\_table tablespaces or general tablespaces. Table compression is enabled using the ROW\_FORMAT=COMPRESSED attribute with CREATE TABLE or ALTER TABLE.

**15.9.1.1** **Overview** **of** **Table** **Compression**

Because processors and cache memories have increased in speed more than disk storage devices, many workloads are disk-bound. Data compression enables smaller database size, reduced I/O, and improved throughput, at the small cost of increased CPU utilization. Compression is especially valuable for read-intensive applications, on systems with enough RAM to keep frequently used data in memory.

An InnoDB table created with ROW\_FORMAT=COMPRESSED can use a smaller page size on disk than the configured [innodb\_page\_size](#_bookmark7) value. Smaller pages require less I/O to read from and write to disk, which is especially valuable for SSD devices.

The compressed page size is specified through the CREATE TABLE or ALTER TABLE KEY\_BLOCK\_SIZE parameter. The different page size requires that the table be placed in a file- per-table tablespace or general tablespace rather than in the system tablespace, as the system tablespace cannot store compressed tables. For more information, see Section 15.6.3.2, “File-Per- Table Tablespaces” , and Section 15.6.3.3, “General Tablespaces” .

The level of compression is the same regardless of the KEY\_BLOCK\_SIZE value. As you specify smaller values for KEY\_BLOCK\_SIZE, you get the I/O benefits of increasingly smaller pages. But if you specify a value that is too small, there is additional overhead to reorganize the pages when data values cannot be compressed enough to fit multiple rows in each page. There is a hard limit on how small KEY\_BLOCK\_SIZE can be for a table, based on the lengths of the key columns for each of its indexes. Specify a value that is too small, and the CREATE TABLE or ALTER TABLE statement fails.

In the buffer pool, the compressed data is held in small pages, with a page size based on the KEY\_BLOCK\_SIZE value. For extracting or updating the column values, MySQL also creates an

uncompressed page in the buffer pool with the uncompressed data. Within the buffer pool, any updates to the uncompressed page are also re-written back to the equivalent compressed page. You might need to size your buffer pool to accommodate the additional data of both compressed and uncompressed pages, although the uncompressed pages are evicted from the buffer pool when space is needed, and then uncompressed again on the next access.

**15.9.1.2** **Creating** **Compressed** **Tables**

Compressed tables can be created in file-per-table tablespaces or in general tablespaces. Table compression is not available for the InnoDB system tablespace. The system tablespace (space 0, the .ibdata files) can contain user-created tables, but it also contains internal system data, which is never compressed. Thus, compression applies only to tables (and indexes) stored in file-per-table or general tablespaces.

**Creating** **a** **Compressed** **Table** **in** **File-Per-Table** **Tablespace**

To create a compressed table in a file-per-table tablespace, [innodb\_file\_per\_table](#_bookmark17) must be enabled (the default). You can set this parameter in the MySQL configuration file (my.cnf or my.ini) or dynamically, using a SET statement.

After the [innodb\_file\_per\_table](#_bookmark17) option is configured, specify the ROW\_FORMAT=COMPRESSED clause or KEY\_BLOCK\_SIZE clause, or both, in a CREATE TABLE or ALTER TABLE statement to create a compressed table in a file-per-table tablespace.

For example, you might use the following statements:

SET GLOBAL innodb\_file\_per\_table=1;

CREATE TABLE t1

(c1 INT PRIMARY KEY)

ROW\_FORMAT=COMPRESSED

KEY\_BLOCK\_SIZE=8;

**Creating** **a** **Compressed** **Table** **in** **a** **General** **Tablespace**

To create a compressed table in a general tablespace, FILE\_BLOCK\_SIZE must be defined for the general tablespace, which is specified when the tablespace is created. The FILE\_BLOCK\_SIZE value must be a valid compressed page size in relation to the [innodb\_page\_size](#_bookmark7) value, and the page size of the compressed table, defined by the CREATE TABLE or ALTER TABLE KEY\_BLOCK\_SIZE clause, must be equal to FILE\_BLOCK\_SIZE/1024. For example, if [innodb\_page\_size=16384](#_bookmark7) and FILE\_BLOCK\_SIZE=8192, the KEY\_BLOCK\_SIZE of the table must be 8. For more information, see Section 15.6.3.3, “General Tablespaces” .

The following example demonstrates creating a general tablespace and adding a compressed table. The example assumes a default [innodb\_page\_size](#_bookmark7) of 16K. The FILE\_BLOCK\_SIZE of 8192 requires that the compressed table have a KEY\_BLOCK\_SIZE of 8.

mysql> **CREATE** **TABLESPACE** **`ts2`** **ADD** **DATAFILE** **'ts2.ibd'** **FILE\_BLOCK\_SIZE** **=** **8192** **Engine=InnoDB;**

mysql> **CREATE** **TABLE** **t4** **(c1** **INT** **PRIMARY** **KEY)** **TABLESPACE** **ts2** **ROW\_FORMAT=COMPRESSED** **KEY\_BLOCK\_SIZE=8;**

**Notes**

• As of MySQL 8.0, the tablespace file for a compressed table is created using the physical page size instead of the InnoDB page size, which makes the initial size of a tablespace file for an empty compressed table smaller than in previous MySQL releases.

• If you specify ROW\_FORMAT=COMPRESSED, you can omit KEY\_BLOCK\_SIZE; the KEY\_BLOCK\_SIZE setting defaults to half the [innodb\_page\_size](#_bookmark7) value.

• If you specify a valid KEY\_BLOCK\_SIZE value, you can omit ROW\_FORMAT=COMPRESSED; compression is enabled automatically.

• To determine the best value for KEY\_BLOCK\_SIZE, typically you create several copies of the same table with different values for this clause, then measure the size of the resulting .ibd files and

see how well each performs with a realistic workload. For general tablespaces, keep in mind that dropping a table does not reduce the size of the general tablespace .ibd file, nor does it return disk space to the operating system. For more information, see Section 15.6.3.3, “General Tablespaces” .

• The KEY\_BLOCK\_SIZE value is treated as a hint; a different size could be used by InnoDB if necessary. For file-per-table tablespaces, the KEY\_BLOCK\_SIZE can only be less than or equal to the [innodb\_page\_size](#_bookmark7) value. If you specify a value greater than the [innodb\_page\_size](#_bookmark7) value, the specified value is ignored, a warning is issued, and KEY\_BLOCK\_SIZE is set to half of the [innodb\_page\_size](#_bookmark7) value. If innodb\_strict\_mode=ON, specifying an invalid KEY\_BLOCK\_SIZE value returns an error. For general tablespaces, valid KEY\_BLOCK\_SIZE values depend on the FILE\_BLOCK\_SIZE setting of the tablespace. For more information, see Section 15.6.3.3, “General Tablespaces” .

• InnoDB supports 32KB and 64KB page sizes but these page sizes do not support compression. For more information, refer to the [innodb\_page\_size](#_bookmark7) documentation.

• The default uncompressed size of InnoDB data pages is 16KB. Depending on the combination of option values, MySQL uses a page size of 1KB, 2KB, 4KB, 8KB, or 16KB for the tablespace data file ( .ibd file). The actual compression algorithm is not affected by the KEY\_BLOCK\_SIZE value; the value determines how large each compressed chunk is, which in turn affects how many rows can be packed into each compressed page.

• When creating a compressed table in a file-per-table tablespace, setting KEY\_BLOCK\_SIZE equal to the InnoDB page size does not typically result in much compression. For example, setting KEY\_BLOCK\_SIZE=16 typically would not result in much compression, since the normal InnoDB page size is 16KB. This setting may still be useful for tables with many long BLOB, VARCHAR or TEXT columns, because such values often do compress well, and might therefore require fewer overflow pages as described in [Section 15.9.1.5, “How Compression Works for InnoDB Tables”](#_bookmark18) . For general tablespaces, a KEY\_BLOCK\_SIZE value equal to the InnoDB page size is not permitted. For more information, see Section 15.6.3.3, “General Tablespaces” .

• All indexes of a table (including the clustered index) are compressed using the same page size, as specified in the CREATE TABLE or ALTER TABLE statement. Table attributes such as ROW\_FORMAT and KEY\_BLOCK\_SIZE are not part of the CREATE INDEX syntax for InnoDB tables, and are ignored if they are specified (although, if specified, they appear in the output of the SHOW CREATE TABLE statement).

• For performance-related configuration options, see [Section 15.9.1.3, “Tuning Compression for](#_bookmark19) [InnoDB Tables”](#_bookmark19) .

**Restrictions** **on** **Compressed** **Tables**

• Compressed tables cannot be stored in the InnoDB system tablespace.

• General tablespaces can contain multiple tables, but compressed and uncompressed tables cannot coexist within the same general tablespace.

• Compression applies to an entire table and all its associated indexes, not to individual rows, despite the clause name ROW\_FORMAT.

• InnoDB does not support compressed temporary tables. When [innodb\_strict\_mode](#_bookmark20) is enabled (the default), CREATE TEMPORARY TABLE returns errors if ROW\_FORMAT=COMPRESSED or KEY\_BLOCK\_SIZE is specified. If [innodb\_strict\_mode](#_bookmark20) is disabled, warnings are issued and the temporary table is created using a non-compressed row format. The same restrictions apply to ALTER TABLE operations on temporary tables.

**15.9.1.3** **Tuning** **Compression** **for** **InnoDB** **Tables**

Most often, the internal optimizations described in [InnoDB Data Storage and Compression](#_bookmark21) ensure that the system runs well with compressed data. However, because the efficiency of compression depends on the nature of your data, you can make decisions that affect the performance of compressed tables:

• Which tables to compress.

• What compressed page size to use.

• Whether to adjust the size of the buffer pool based on run-time performance characteristics, such as the amount of time the system spends compressing and uncompressing data. Whether the workload is more like a data warehouse (primarily queries) or an OLTP system (mix of queries and DML).

• If the system performs DML operations on compressed tables, and the way the data is distributed leads to expensive compression failures at runtime, you might adjust additional advanced configuration options.

Use the guidelines in this section to help make those architectural and configuration choices. When you are ready to conduct long-term testing and put compressed tables into production, see [Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”](#_bookmark22) for ways to verify the effectiveness of those choices under real-world conditions.

**When** **to** **Use** **Compression**

In general, compression works best on tables that include a reasonable number of character string columns and where the data is read far more often than it is written. Because there are no guaranteed ways to predict whether or not compression benefits a particular situation, always test with a specific workload and data set running on a representative configuration. Consider the following factors when deciding which tables to compress.

**Data** **Characteristics** **and** **Compression**

A key determinant of the efficiency of compression in reducing the size of data files is the nature of the data itself. Recall that compression works by identifying repeated strings of bytes in a block of data. Completely randomized data is the worst case. Typical data often has repeated values, and so compresses effectively. Character strings often compress well, whether defined in CHAR, VARCHAR, TEXT or BLOB columns. On the other hand, tables containing mostly binary data (integers or floating point numbers) or data that is previously compressed (for example JPEG or PNG images) may not generally compress well, significantly or at all.

You choose whether to turn on compression for each InnoDB table. A table and all of its indexes use the same (compressed) page size. It might be that the primary key (clustered) index, which contains the data for all columns of a table, compresses more effectively than the secondary indexes. For those cases where there are long rows, the use of compression might result in long column values being stored “off-page” , as discussed in [DYNAMIC Row Format](#_bookmark23). Those overflow pages may compress well. Given these considerations, for many applications, some tables compress more effectively than others, and you might find that your workload performs best only with a subset of tables compressed.

To determine whether or not to compress a particular table, conduct experiments. You can get a rough estimate of how efficiently your data can be compressed by using a utility that implements LZ77 compression (such as gzip or WinZip) on a copy of the .ibd file for an uncompressed table. You can expect less compression from a MySQL compressed table than from file-based compression tools, because MySQL compresses data in chunks based on the page size, 16KB by default. In addition to user data, the page format includes some internal system data that is not compressed. File-based compression utilities can examine much larger chunks of data, and so might find more repeated strings in a huge file than MySQL can find in an individual page.

Another way to test compression on a specific table is to copy some data from your uncompressed table to a similar, compressed table (having all the same indexes) in a file-per-table tablespace and look at the size of the resulting .ibd file. For example:

USE test;

SET GLOBAL innodb\_file\_per\_table=1;

SET GLOBAL autocommit=0;

-- Create an uncompressed table with a million or two rows.

CREATE TABLE big\_table AS SELECT \* FROM information\_schema.columns;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

INSERT INTO big\_table SELECT \* FROM big\_table;

COMMIT;

ALTER TABLE big\_table ADD id int unsigned NOT NULL PRIMARY KEY auto\_increment;

SHOW CREATE TABLE big\_table\G

select count(id) from big\_table;

-- Check how much space is needed for the uncompressed table.

\! ls -l data/test/big\_table.ibd

CREATE TABLE key\_block\_size\_4 LIKE big\_table;

ALTER TABLE key\_block\_size\_4 key\_block\_size=4 row\_format=compressed;

INSERT INTO key\_block\_size\_4 SELECT \* FROM big\_table;

commit;

-- Check how much space is needed for a compressed table

-- with particular compression settings .

\! ls -l data/test/key\_block\_size\_4.ibd

This experiment produced the following numbers, which of course could vary considerably depending on your table structure and data:

-rw-rw---- 1 cirrus staff 310378496 Jan 9 13:44 data/test/big\_table.ibd

-rw-rw---- 1 cirrus staff 83886080 Jan 9 15:10 data/test/key\_block\_size\_4.ibd

To see whether compression is efficient for your particular workload:

• For simple tests, use a MySQL instance with no other compressed tables and run queries against the Information Schema INNODB\_CMP table.

• For more elaborate tests involving workloads with multiple compressed tables, run queries against the Information Schema INNODB\_CMP\_PER\_INDEX table. Because the statistics in the

INNODB\_CMP\_PER\_INDEX table are expensive to collect, you must enable the configuration option [innodb\_cmp\_per\_index\_enabled](#_bookmark24) before querying that table, and you might restrict such testing to a development server or a non-critical replica server.

• Run some typical SQL statements against the compressed table you are testing.

• Examine the ratio of successful compression operations to overall

compression operations by querying INFORMATION\_SCHEMA.INNODB\_CMP or INFORMATION\_SCHEMA.INNODB\_CMP\_PER\_INDEX, and comparing COMPRESS\_OPS to

COMPRESS\_OPS\_OK.

• If a high percentage of compression operations complete successfully, the table might be a good candidate for compression.

• If you get a high proportion of compression failures, you can adjust [innodb\_compression\_level](#_bookmark25), [innodb\_compression\_failure\_threshold\_pct](#_bookmark26), and [innodb\_compression\_pad\_pct\_max](#_bookmark27) options as described in [Section 15.9.1.6, “Compression for OLTP Workloads”](#_bookmark28) , and try further tests.

**Database** **Compression** **versus** **Application** **Compression**

Decide whether to compress data in your application or in the table; do not use both types of compression for the same data. When you compress the data in the application and store the results in a compressed table, extra space savings are extremely unlikely, and the double compression just wastes CPU cycles.

**Compressing** **in** **the** **Database**

When enabled, MySQL table compression is automatic and applies to all columns and index values. The columns can still be tested with operators such as LIKE, and sort operations can still use indexes even when the index values are compressed. Because indexes are often a significant fraction of the total size of a database, compression could result in significant savings in storage, I/O or processor time. The compression and decompression operations happen on the database server, which likely is a powerful system that is sized to handle the expected load.

**Compressing** **in** **the** **Application**

If you compress data such as text in your application, before it is inserted into the database, You might save overhead for data that does not compress well by compressing some columns and not others. This approach uses CPU cycles for compression and uncompression on the client machine rather than the database server, which might be appropriate for a distributed application with many clients, or where the client machine has spare CPU cycles.

**Hybrid** **Approach**

Of course, it is possible to combine these approaches. For some applications, it may be appropriate to use some compressed tables and some uncompressed tables. It may be best to externally compress some data (and store it in uncompressed tables) and allow MySQL to compress (some of) the other tables in the application. As always, up-front design and real-life testing are valuable in reaching the right decision.

**Workload** **Characteristics** **and** **Compression**

In addition to choosing which tables to compress (and the page size), the workload is another key determinant of performance. If the application is dominated by reads, rather than updates, fewer pages need to be reorganized and recompressed after the index page runs out of room for the per- page “modification log” that MySQL maintains for compressed data. If the updates predominantly change non-indexed columns or those containing BLOBs or large strings that happen to be stored “off- page” , the overhead of compression may be acceptable. If the only changes to a table are INSERTs that use a monotonically increasing primary key, and there are few secondary indexes, there is little need to reorganize and recompress index pages. Since MySQL can “delete-mark” and delete rows on compressed pages “in place” by modifying uncompressed data, DELETE operations on a table are relatively efficient.

For some environments, the time it takes to load data can be as important as run-time retrieval. Especially in data warehouse environments, many tables may be read-only or read-mostly. In those cases, it might or might not be acceptable to pay the price of compression in terms of increased load time, unless the resulting savings in fewer disk reads or in storage cost is significant.

Fundamentally, compression works best when the CPU time is available for compressing and uncompressing data. Thus, if your workload is I/O bound, rather than CPU-bound, you might find that compression can improve overall performance. When you test your application performance with different compression configurations, test on a platform similar to the planned configuration of the production system.

**Configuration** **Characteristics** **and** **Compression**

Reading and writing database pages from and to disk is the slowest aspect of system performance. Compression attempts to reduce I/O by using CPU time to compress and uncompress data, and is most effective when I/O is a relatively scarce resource compared to processor cycles.

This is often especially the case when running in a multi-user environment with fast, multi-core CPUs. When a page of a compressed table is in memory, MySQL often uses additional memory, typically 16KB, in the buffer pool for an uncompressed copy of the page. The adaptive LRU algorithm attempts to balance the use of memory between compressed and uncompressed pages to take into account

whether the workload is running in an I/O-bound or CPU-bound manner. Still, a configuration with more memory dedicated to the buffer pool tends to run better when using compressed tables than a configuration where memory is highly constrained.

**Choosing** **the** **Compressed** **Page** **Size**

The optimal setting of the compressed page size depends on the type and distribution of data that the table and its indexes contain. The compressed page size should always be bigger than the maximum record size, or operations may fail as noted in [Compression of B-Tree Pages](#_bookmark29).

Setting the compressed page size too large wastes some space, but the pages do not have to be compressed as often. If the compressed page size is set too small, inserts or updates may require time-consuming recompression, and the B-tree nodes may have to be split more frequently, leading to bigger data files and less efficient indexing.

Typically, you set the compressed page size to 8K or 4K bytes. Given that the maximum row size for an InnoDB table is around 8K, KEY\_BLOCK\_SIZE=8 is usually a safe choice.

**15.9.1.4** **Monitoring** **InnoDB** **Table** **Compression** **at** **Runtime**

Overall application performance, CPU and I/O utilization and the size of disk files are good indicators of how effective compression is for your application. This section builds on the performance tuning advice from [Section 15.9.1.3, “Tuning Compression for InnoDB Tables”](#_bookmark19) , and shows how to find problems that might not turn up during initial testing.

To dig deeper into performance considerations for compressed tables, you can monitor compression performance at runtime using the Information Schema tables described in [Example 15.1, “Using the](#_bookmark30) [Compression Information Schema Tables”](#_bookmark30) . These tables reflect the internal use of memory and the rates of compression used overall.

The INNODB\_CMP table reports information about compression activity for each compressed page size (KEY\_BLOCK\_SIZE) in use. The information in these tables is system-wide: it summarizes the compression statistics across all compressed tables in your database. You can use this data to help decide whether or not to compress a table by examining these tables when no other compressed tables are being accessed. It involves relatively low overhead on the server, so you might query it periodically on a production server to check the overall efficiency of the compression feature.

The INNODB\_CMP\_PER\_INDEX table reports information about compression activity for individual tables and indexes. This information is more targeted and more useful for evaluating compression efficiency and diagnosing performance issues one table or index at a time. (Because that each InnoDB table is represented as a clustered index, MySQL does not make a big distinction between tables and indexes in this context.) The INNODB\_CMP\_PER\_INDEX table does involve substantial overhead, so it is more suitable for development servers, where you can compare the effects of different workloads, data, and compression settings in isolation. To guard against imposing this monitoring overhead by accident, you must enable the [innodb\_cmp\_per\_index\_enabled](#_bookmark24) configuration option before you can query the INNODB\_CMP\_PER\_INDEX table.

The key statistics to consider are the number of, and amount of time spent performing, compression and uncompression operations. Since MySQL splits B-tree nodes when they are too full to contain the compressed data following a modification, compare the number of “successful” compression operations with the number of such operations overall. Based on the information in the INNODB\_CMP and INNODB\_CMP\_PER\_INDEX tables and overall application performance and hardware resource utilization, you might make changes in your hardware configuration, adjust the size of the buffer pool, choose a different page size, or select a different set of tables to compress.

If the amount of CPU time required for compressing and uncompressing is high, changing to faster or multi-core CPUs can help improve performance with the same data, application workload and set of compressed tables. Increasing the size of the buffer pool might also help performance, so that more uncompressed pages can stay in memory, reducing the need to uncompress pages that exist in memory only in compressed form.



A large number of compression operations overall (compared to the number of INSERT, UPDATE and DELETE operations in your application and the size of the database) could indicate that some of your compressed tables are being updated too heavily for effective compression. If so, choose a larger page size, or be more selective about which tables you compress.

If the number of “successful” compression operations (COMPRESS\_OPS\_OK) is a high percentage of the total number of compression operations (COMPRESS\_OPS), then the system is likely performing well. If the ratio is low, then MySQL is reorganizing, recompressing, and splitting B-tree nodes more often than is desirable. In this case, avoid compressing some tables, or increase KEY\_BLOCK\_SIZE for some of the compressed tables. You might turn off compression for tables that cause the number of “compression failures” in your application to be more than 1% or 2% of the total. (Such a failure ratio might be acceptable during a temporary operation such as a data load).

**15.9.1.5** **How** **Compression** **Works** **for** **InnoDB** **Tables**

This section describes some internal implementation details about compression for InnoDB tables. The information presented here may be helpful in tuning for performance, but is not necessary to know for basic use of compression.

**Compression** **Algorithms**

Some operating systems implement compression at the file system level. Files are typically divided into fixed-size blocks that are compressed into variable-size blocks, which easily leads into fragmentation. Every time something inside a block is modified, the whole block is recompressed before it is written to disk. These properties make this compression technique unsuitable for use in an update-intensive database system.

MySQL implements compression with the help of the well-known [zlib library](http://www.zlib.net/), which implements the LZ77 compression algorithm. This compression algorithm is mature, robust, and efficient in both CPU utilization and in reduction of data size. The algorithm is “lossless” , so that the original uncompressed data can always be reconstructed from the compressed form. LZ77 compression works by finding sequences of data that are repeated within the data to be compressed. The patterns of values in your data determine how well it compresses, but typical user data often compresses by 50% or more.

**Note**

InnoDB supports the zlib library up to version 1.2.11, which is the version bundled with MySQL 8.0.

Unlike compression performed by an application, or compression features of some other database management systems, InnoDB compression applies both to user data and to indexes. In many cases, indexes can constitute 40-50% or more of the total database size, so this difference is significant. When compression is working well for a data set, the size of the InnoDB data files (the file-per-table tablespace or general tablespace .ibd files) is 25% to 50% of the uncompressed

size or possibly smaller. Depending on the workload, this smaller database can in turn lead to a reduction in I/O, and an increase in throughput, at a modest cost in terms of increased CPU utilization. You can adjust the balance between compression level and CPU overhead by modifying the [innodb\_compression\_level](#_bookmark25) configuration option.

**InnoDB** **Data** **Storage** **and** **Compression**

All user data in InnoDB tables is stored in pages comprising a B-tree index (the clustered index). In some other database systems, this type of index is called an “index-organized table” . Each row in the index node contains the values of the (user-specified or system-generated) primary key and all the other columns of the table.

Secondary indexes in InnoDB tables are also B-trees, containing pairs of values: the index key and a pointer to a row in the clustered index. The pointer is in fact the value of the primary key of the table, which is used to access the clustered index if columns other than the index key and primary key are required. Secondary index records must always fit on a single B-tree page.

The compression of B-tree nodes (of both clustered and secondary indexes) is handled differently from compression of overflow pages used to store long VARCHAR, BLOB, or TEXT columns, as explained in the following sections.

**Compression** **of** **B-Tree** **Pages**

Because they are frequently updated, B-tree pages require special treatment. It is important to minimize the number of times B-tree nodes are split, as well as to minimize the need to uncompress and recompress their content.

One technique MySQL uses is to maintain some system information in the B-tree node in uncompressed form, thus facilitating certain in-place updates. For example, this allows rows to be delete-marked and deleted without any compression operation.

In addition, MySQL attempts to avoid unnecessary uncompression and recompression of index pages when they are changed. Within each B-tree page, the system keeps an uncompressed “modification log” to record changes made to the page. Updates and inserts of small records may be written to this modification log without requiring the entire page to be completely reconstructed.

When the space for the modification log runs out, InnoDB uncompresses the page, applies the changes and recompresses the page. If recompression fails (a situation known as a compression failure), the B-tree nodes are split and the process is repeated until the update or insert succeeds.

To avoid frequent compression failures in write-intensive workloads, such as for OLTP applications, MySQL sometimes reserves some empty space (padding) in the page, so that the modification log fills up sooner and the page is recompressed while there is still enough room to avoid splitting it. The amount of padding space left in each page varies as the system keeps track of the frequency of page splits. On a busy server doing frequent writes to compressed tables, you can adjust the [innodb\_compression\_failure\_threshold\_pct](#_bookmark26), and [innodb\_compression\_pad\_pct\_max](#_bookmark27) configuration options to fine-tune this mechanism.

Generally, MySQL requires that each B-tree page in an InnoDB table can accommodate at least two records. For compressed tables, this requirement has been relaxed. Leaf pages of B-tree nodes (whether of the primary key or secondary indexes) only need to accommodate one record, but that record must fit, in uncompressed form, in the per-page modification log. If [innodb\_strict\_mode](#_bookmark20) is ON, MySQL checks the maximum row size during CREATE TABLE or CREATE INDEX. If the row does not fit, the following error message is issued: ERROR HY000: Too big row.

If you create a table when [innodb\_strict\_mode](#_bookmark20) is OFF, and a subsequent INSERT or UPDATE statement attempts to create an index entry that does not fit in the size of the compressed page, the operation fails with ERROR 42000: Row size too large. (This error message does not name the index for which the record is too large, or mention the length of the index record or the maximum record size on that particular index page.) To solve this problem, rebuild the table with ALTER TABLE and select a larger compressed page size (KEY\_BLOCK\_SIZE), shorten any column prefix indexes, or disable compression entirely with ROW\_FORMAT=DYNAMIC or ROW\_FORMAT=COMPACT.

[innodb\_strict\_mode](#_bookmark20) is not applicable to general tablespaces, which also support compressed tables. Tablespace management rules for general tablespaces are strictly enforced independently of [innodb\_strict\_mode](#_bookmark20). For more information, see Section 13.1.21, “CREATE TABLESPACE

Statement” .

**Compressing** **BLOB,** **VARCHAR,** **and** **TEXT** **Columns**

In an InnoDB table, BLOB, VARCHAR, and TEXT columns that are not part of the primary key may be stored on separately allocated overflow pages. We refer to these columns as off-page columns. Their values are stored on singly-linked lists of overflow pages.

For tables created in ROW\_FORMAT=DYNAMIC or ROW\_FORMAT=COMPRESSED, the values of BLOB, TEXT, or VARCHAR columns may be stored fully off-page, depending on their length and the length of the entire row. For columns that are stored off-page, the clustered index record only contains 20-byte pointers to the overflow pages, one per column. Whether any columns are stored off-page depends



on the page size and the total size of the row. When the row is too long to fit entirely within the page of the clustered index, MySQL chooses the longest columns for off-page storage until the row fits on the clustered index page. As noted above, if a row does not fit by itself on a compressed page, an error occurs.

**Note**

For tables created in ROW\_FORMAT=DYNAMIC or ROW\_FORMAT=COMPRESSED, TEXT and BLOB columns that are less than or equal to 40 bytes are always stored in-line.

Tables that use ROW\_FORMAT=REDUNDANT and ROW\_FORMAT=COMPACT store the first 768 bytes of BLOB, VARCHAR, and TEXT columns in the clustered index record along with the primary key. The 768- byte prefix is followed by a 20-byte pointer to the overflow pages that contain the rest of the column value.

When a table is in COMPRESSED format, all data written to overflow pages is compressed “as is”; that is, MySQL applies the zlib compression algorithm to the entire data item. Other than the data, compressed overflow pages contain an uncompressed header and trailer comprising a page checksum and a link to the next overflow page, among other things. Therefore, very significant storage savings can be obtained for longer BLOB, TEXT, or VARCHAR columns if the data is highly compressible, as is often the case with text data. Image data, such as JPEG, is typically already compressed and so does not benefit much from being stored in a compressed table; the double compression can waste CPU cycles for little or no space savings.

The overflow pages are of the same size as other pages. A row containing ten columns stored off- page occupies ten overflow pages, even if the total length of the columns is only 8K bytes. In an uncompressed table, ten uncompressed overflow pages occupy 160K bytes. In a compressed table with an 8K page size, they occupy only 80K bytes. Thus, it is often more efficient to use compressed table format for tables with long column values.

For file-per-table tablespaces, using a 16K compressed page size can reduce storage and I/O

costs for BLOB, VARCHAR, or TEXT columns, because such data often compress well, and might therefore require fewer overflow pages, even though the B-tree nodes themselves take as many pages as in the uncompressed form. General tablespaces do not support a 16K compressed page size (KEY\_BLOCK\_SIZE). For more information, see Section 15.6.3.3, “General Tablespaces” .

**Compression** **and** **the** **InnoDB** **Buffer** **Pool**

In a compressed InnoDB table, every compressed page (whether 1K, 2K, 4K or 8K) corresponds to an uncompressed page of 16K bytes (or a smaller size if [innodb\_page\_size](#_bookmark7) is set). To access the data in a page, MySQL reads the compressed page from disk if it is not already in the buffer pool, then uncompresses the page to its original form. This section describes how InnoDB manages the buffer pool with respect to pages of compressed tables.

To minimize I/O and to reduce the need to uncompress a page, at times the buffer pool contains

both the compressed and uncompressed form of a database page. To make room for other required database pages, MySQL can evict from the buffer pool an uncompressed page, while leaving the compressed page in memory. Or, if a page has not been accessed in a while, the compressed form of the page might be written to disk, to free space for other data. Thus, at any given time, the buffer pool might contain both the compressed and uncompressed forms of the page, or only the compressed form of the page, or neither.

MySQL keeps track of which pages to keep in memory and which to evict using a least-recently- used (LRU) list, so that hot (frequently accessed) data tends to stay in memory. When compressed tables are accessed, MySQL uses an adaptive LRU algorithm to achieve an appropriate balance of compressed and uncompressed pages in memory. This adaptive algorithm is sensitive to whether the system is running in an I/O-bound or CPU-bound manner. The goal is to avoid spending too much processing time uncompressing pages when the CPU is busy, and to avoid doing excess I/O when the CPU has spare cycles that can be used for uncompressing compressed pages (that may already be

in memory). When the system is I/O-bound, the algorithm prefers to evict the uncompressed copy of a page rather than both copies, to make more room for other disk pages to become memory resident. When the system is CPU-bound, MySQL prefers to evict both the compressed and uncompressed page, so that more memory can be used for “hot” pages and reducing the need to uncompress data in memory only in compressed form.

**Compression** **and** **the** **InnoDB** **Redo** **Log** **Files**

Before a compressed page is written to a data file, MySQL writes a copy of the page to the redo log (if it has been recompressed since the last time it was written to the database). This is done to ensure that redo logs are usable for crash recovery, even in the unlikely case that the zlib library is upgraded and that change introduces a compatibility problem with the compressed data. Therefore, some increase in the size of log files, or a need for more frequent checkpoints, can be expected when using compression. The amount of increase in the log file size or checkpoint frequency depends on the number of times compressed pages are modified in a way that requires reorganization and recompression.

To create a compressed table in a file-per-table tablespace, [innodb\_file\_per\_table](#_bookmark17) must be enabled. There is no dependence on the [innodb\_file\_per\_table](#_bookmark17) setting when creating a compressed table in a general tablespace. For more information, see Section 15.6.3.3, “General Tablespaces” .

**15.9.1.6** **Compression** **for** **OLTP** **Workloads**

Traditionally, the InnoDB compression feature was recommended primarily for read-only or read- mostly workloads, such as in a data warehouse configuration. The rise of SSD storage devices, which are fast but relatively small and expensive, makes compression attractive also for OLTP workloads: high-traffic, interactive websites can reduce their storage requirements and their I/O operations per second (IOPS) by using compressed tables with applications that do frequent INSERT, UPDATE, and DELETE operations.

These configuration options let you adjust the way compression works for a particular MySQL instance, with an emphasis on performance and scalability for write-intensive operations:

• [innodb\_compression\_level](#_bookmark25) lets you turn the degree of compression up or down. 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.

• [innodb\_compression\_failure\_threshold\_pct](#_bookmark26) specifies a cutoff point for compression failures during updates to a compressed table. 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](#_bookmark27)

• [innodb\_compression\_pad\_pct\_max](#_bookmark27) lets you adjust the maximum amount of space reserved within each page to record changes to compressed rows, without needing to compress the entire page again. The higher the value, the more changes can be recorded without recompressing the page. MySQL uses a variable amount of free space for the pages within each compressed table, only when a designated percentage of compression operations “fail” at runtime, requiring an expensive operation to split the compressed page.

• [innodb\_log\_compressed\_pages](#_bookmark31) lets you disable writing of images of re-compressed pages to the redo log. Re-compression may occur when changes are made to compressed data. This option 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, disable [innodb\_log\_compressed\_pages](#_bookmark31) to reduce redo log generation for workloads that modify compressed data.

Because working with compressed data sometimes involves keeping both compressed and uncompressed versions of a page in memory at the same time, when using compression with an



OLTP-style workload, be prepared to increase the value of the [innodb\_buffer\_pool\_size](#_bookmark14) configuration option.

**15.9.1.7** **SQL** **Compression** **Syntax** **Warnings** **and** **Errors**

This section describes syntax warnings and errors that you may encounter when using the table compression feature with file-per-table tablespaces and general tablespaces.

**SQL** **Compression** **Syntax** **Warnings** **and** **Errors** **for** **File-Per-Table** **Tablespaces**

When [innodb\_strict\_mode](#_bookmark20) is enabled (the default), specifying ROW\_FORMAT=COMPRESSED or KEY\_BLOCK\_SIZE in CREATE TABLE or ALTER TABLE statements produces the following error if [innodb\_file\_per\_table](#_bookmark17) is disabled.

ERROR 1031 (HY000): Table storage engine for 't1' doesn't have this option

**Note**

The table is not created if the current configuration does not permit using compressed tables.

When [innodb\_strict\_mode](#_bookmark20) is disabled, specifying ROW\_FORMAT=COMPRESSED or KEY\_BLOCK\_SIZE in CREATE TABLE or ALTER TABLE statements produces the following warnings if [innodb\_file\_per\_table](#_bookmark17) is disabled.

mysql> **SHOW** **WARNINGS;**

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

| Level | Code | Message |

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

| Warning | 1478 | InnoDB: KEY\_BLOCK\_SIZE requires innodb\_file\_per\_table . |

| Warning | 1478 | InnoDB: ignoring KEY\_BLOCK\_SIZE=4 . |

| Warning | 1478 | InnoDB: ROW\_FORMAT=COMPRESSED requires innodb\_file\_per\_table . |

| Warning | 1478 | InnoDB: assuming ROW\_FORMAT=DYNAMIC . |

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

**Note**

These messages are only warnings, not errors, and the table is created without compression, as if the options were not specified.

The “non-strict” behavior lets you import a mysqldump file into a database that does not support compressed tables, even if the source database contained compressed tables. In that case, MySQL creates the table in ROW\_FORMAT=DYNAMIC instead of preventing the operation.

To import the dump file into a new database, and have the tables re-created as they exist in the original database, ensure the server has the proper setting for the [innodb\_file\_per\_table](#_bookmark17) configuration parameter.

The attribute KEY\_BLOCK\_SIZE is permitted only when ROW\_FORMAT is specified as COMPRESSED or is omitted. Specifying a KEY\_BLOCK\_SIZE with any other ROW\_FORMAT generates a warning that you can view with SHOW WARNINGS. However, the table is non-compressed; the specified KEY\_BLOCK\_SIZE is ignored).

|  |  |  |
| --- | --- | --- |
| **Level** | **Code** | **Message** |
| Warning | 1478 | InnoDB: ignoring KEY\_BLOCK\_SIZE=*n* unless ROW\_FORMAT=COMPRESSED. |

If you are running with [innodb\_strict\_mode](#_bookmark20) enabled, the combination of a KEY\_BLOCK\_SIZE with

any ROW\_FORMAT other than COMPRESSED generates an error, not a warning, and the table is not created.

[Table 15.12, “ROW\_FORMAT and KEY\_BLOCK\_SIZE Options”](#_bookmark32) provides an overview the ROW\_FORMAT and KEY\_BLOCK\_SIZE options that are used with CREATE TABLE or ALTER TABLE.

**Table** **15.12** **ROW\_FORMAT** **and** **KEY\_BLOCK\_SIZE** **Options**

|  |  |  |
| --- | --- | --- |
| **Option** | **Usage** **Notes** | **Description** |
| ROW\_FORMAT=REDUNDANT | Storage format used prior to MySQL 5.0.3 | Less efficient than ROW\_FORMAT=COMPACT; for backward compatibility |
| ROW\_FORMAT=COMPACT | Default storage format since MySQL 5.0.3 | Stores a prefix of 768 bytes of long column values in the clustered index page, with the remaining bytes stored in an overflow page |
| ROW\_FORMAT=DYNAMIC |  | Store values within the clustered index page if they fit; if not, stores only a 20-byte pointer to an overflow page (no prefix) |
| ROW\_FORMAT=COMPRESSED |  | Compresses the table and  indexes using zlib |
| KEY\_BLOCK\_SIZE=*n* |  | Specifies compressed  page size of 1, 2, 4, 8  or 16 kilobytes; implies  ROW\_FORMAT=COMPRESSED.  For general tablespaces, a KEY\_BLOCK\_SIZE value equal to the InnoDB page size is not permitted. |

[Table 15.13, “CREATE/ALTER TABLE Warnings and Errors when InnoDB Strict Mode is OFF”](#_bookmark33)

summarizes error conditions that occur with certain combinations of configuration parameters and options on the CREATE TABLE or ALTER TABLE statements, and how the options appear in the output of SHOW TABLE STATUS.

When [innodb\_strict\_mode](#_bookmark20) is OFF, MySQL creates or alters the table, but ignores certain settings as shown below. You can see the warning messages in the MySQL error log. When [innodb\_strict\_mode](#_bookmark20) is ON, these specified combinations of options generate errors, and the table is not created or altered. To see the full description of the error condition, issue the SHOW ERRORS statement: example:

mysql> **CREATE** **TABLE** **x** **(id** **INT** **PRIMARY** **KEY,** **c** **INT)**

-> **ENGINE=INNODB** **KEY\_BLOCK\_SIZE=33333;**

ERROR 1005 (HY000): Can't create table 'test.x' (errno: 1478)

mysql> **SHOW** **ERRORS;**

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

| Level | Code | Message |

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

| Error | 1478 | InnoDB: invalid KEY\_BLOCK\_SIZE=33333 . |

| Error | 1005 | Can't create table 'test .x' (errno: 1478) |

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

**Table** **15.13** **CREATE/ALTER** **TABLE** **Warnings** **and** **Errors** **when** **InnoDB** **Strict** **Mode** **is** **OFF**

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Warning** **or** **Error** **Condition** | **Resulting** **ROW\_FORMAT,** **as** **shown** **in** **SHOW** **TABLE**  **STATUS** |
| ROW\_FORMAT=REDUNDANT | None | REDUNDANT |

**KEY\_BLOCK\_SIZE=8;**

ERROR 1478 (HY000): InnoDB: Tablespace `ts1` cannot contain a COMPRESSED table

**KEY\_BLOCK\_SIZE=4;**

ERROR 1478 (HY000): InnoDB: Tablespace `ts2` uses block size 8192 and cannot

contain a table with physical page size 4096

|  |  |  |
| --- | --- | --- |
| **Syntax** | **Warning** **or** **Error** **Condition** | **Resulting** **ROW\_FORMAT,** **as** **shown** **in** **SHOW** **TABLE**  **STATUS** |
| ROW\_FORMAT=COMPACT | None | COMPACT |
| ROW\_FORMAT=COMPRESSED  or ROW\_FORMAT=DYNAMIC or KEY\_BLOCK\_SIZE is specified | Ignored for file-per-  table tablespaces unless [innodb\_file\_per\_table](#_bookmark17) is enabled. General tablespaces support all row formats. See  Section 15.6.3.3, “General  Tablespaces” . | the default row format  for file-per-table  tablespaces; the  specified row format for  general tablespaces |
| Invalid KEY\_BLOCK\_SIZE is specified (not 1, 2, 4, 8 or 16) | KEY\_BLOCK\_SIZE is ignored | the specified row format, or the default row format |
| ROW\_FORMAT=COMPRESSED  and valid KEY\_BLOCK\_SIZE are specified | None; KEY\_BLOCK\_SIZE  specified is used | COMPRESSED |
| KEY\_BLOCK\_SIZE is specified with REDUNDANT, COMPACT or DYNAMIC row format | KEY\_BLOCK\_SIZE is ignored | REDUNDANT, COMPACT or  DYNAMIC |
| ROW\_FORMAT is not one  of REDUNDANT, COMPACT,  DYNAMIC or COMPRESSED | Ignored if recognized by the MySQL parser. Otherwise, an error is issued. | the default row format or N/A |

When innodb\_strict\_mode is ON, MySQL rejects invalid ROW\_FORMAT or KEY\_BLOCK\_SIZE parameters and issues errors. Strict mode is ON by default. When innodb\_strict\_mode is OFF, MySQL issues warnings instead of errors for ignored invalid parameters.

It is not possible to see the chosen KEY\_BLOCK\_SIZE using SHOW TABLE STATUS. The statement SHOW CREATE TABLE displays the KEY\_BLOCK\_SIZE (even if it was ignored when creating the table). The real compressed page size of the table cannot be displayed by MySQL.

**SQL** **Compression** **Syntax** **Warnings** **and** **Errors** **for** **General** **Tablespaces**

• If FILE\_BLOCK\_SIZE was not defined for the general tablespace when the tablespace was created, the tablespace cannot contain compressed tables. If you attempt to add a compressed table, an error is returned, as shown in the following example:

mysql> **CREATE** **TABLESPACE** **`ts1`** **ADD** **DATAFILE** **'ts1.ibd'** **Engine=InnoDB;**

mysql> **CREATE** **TABLE** **t1** **(c1** **INT** **PRIMARY** **KEY)** **TABLESPACE** **ts1** **ROW\_FORMAT=COMPRESSED**

• Attempting to add a table with an invalid KEY\_BLOCK\_SIZE to a general tablespace returns an error, as shown in the following example:

mysql> **CREATE** **TABLESPACE** **`ts2`** **ADD** **DATAFILE** **'ts2.ibd'** **FILE\_BLOCK\_SIZE** **=** **8192** **Engine=InnoDB;**

mysql> **CREATE** **TABLE** **t2** **(c1** **INT** **PRIMARY** **KEY)** **TABLESPACE** **ts2** **ROW\_FORMAT=COMPRESSED**

For general tablespaces, the KEY\_BLOCK\_SIZE of the table must be equal to the FILE\_BLOCK\_SIZE of the tablespace divided by 1024. For example, if the FILE\_BLOCK\_SIZE of the tablespace is 8192, the KEY\_BLOCK\_SIZE of the table must be 8.

• Attempting to add a table with an uncompressed row format to a general tablespace configured to store compressed tables returns an error, as shown in the following example:



mysql> **CREATE** **TABLESPACE** **`ts3`** **ADD** **DATAFILE** **'ts3.ibd'** **FILE\_BLOCK\_SIZE** **=** **8192** **Engine=InnoDB;**

mysql> **CREATE** **TABLE** **t3** **(c1** **INT** **PRIMARY** **KEY)** **TABLESPACE** **ts3** **ROW\_FORMAT=COMPACT;**

ERROR 1478 (HY000): InnoDB: Tablespace `ts3` uses block size 8192 and cannot

contain a table with physical page size 16384

[innodb\_strict\_mode](#_bookmark20) is not applicable to general tablespaces. Tablespace management rules for general tablespaces are strictly enforced independently of [innodb\_strict\_mode](#_bookmark20). For more information, see Section 13.1.21, “CREATE TABLESPACE Statement” .

For more information about using compressed tables with general tablespaces, see Section 15.6.3.3, “General Tablespaces” .

**15.9.2** **InnoDB** **Page** **Compression**

InnoDB supports page-level compression for tables that reside in file-per-table tablespaces. This feature is referred to as *Transparent* *Page* *Compression*. Page compression is enabled by specifying the COMPRESSION attribute with CREATE TABLE or ALTER TABLE. Supported compression algorithms include Zlib and LZ4.

**Supported** **Platforms**

Page compression requires sparse file and hole punching support. Page compression is supported on Windows with NTFS, and on the following subset of MySQL-supported Linux platforms where the kernel level provides hole punching support:

• RHEL 7 and derived distributions that use kernel version 3.10.0-123 or higher

• OEL 5.10 (UEK2) kernel version 2.6.39 or higher

• OEL 6.5 (UEK3) kernel version 3.8.13 or higher

• OEL 7.0 kernel version 3.8.13 or higher

• SLE11 kernel version 3.0-x

• SLE12 kernel version 3.12-x

• OES11 kernel version 3.0-x

• Ubuntu 14.0.4 LTS kernel version 3.13 or higher

• Ubuntu 12.0.4 LTS kernel version 3.2 or higher

• Debian 7 kernel version 3.2 or higher

**Note**

All of the available file systems for a given Linux distribution may not support hole punching.

**How** **Page** **Compression** **Works**

When a page is written, it is compressed using the specified compression algorithm. The compressed data is written to disk, where the hole punching mechanism releases empty blocks from the end of the page. If compression fails, data is written out as-is.

**Hole** **Punch** **Size** **on** **Linux**

On Linux systems, the file system block size is the unit size used for hole punching. Therefore, page compression only works if page data can be compressed to a size that is less than or equal to the

InnoDB page size minus the file system block size. For example, if [innodb\_page\_size=16K](#_bookmark7) and the file system block size is 4K, page data must compress to less than or equal to 12K to make hole punching possible.

**Hole** **Punch** **Size** **on** **Windows**

On Windows systems, the underlying infrastructure for sparse files is based on NTFS compression. Hole punching size is the NTFS compression unit, which is 16 times the NTFS cluster size. Cluster sizes and their compression units are shown in the following table:

**Table** **15.14** **Windows** **NTFS** **Cluster** **Size** **and** **Compression** **Units**

|  |  |
| --- | --- |
| **Cluster** **Size** | **Compression** **Unit** |
| 512 Bytes | 8 KB |
| 1 KB | 16 KB |
| 2 KB | 32 KB |
| 4 KB | 64 KB |

Page compression on Windows systems only works if page data can be compressed to a size that is less than or equal to the InnoDB page size minus the compression unit size.

The default NTFS cluster size is 4KB, for which the compression unit size is 64KB. This means that page compression has no benefit for an out-of-the box Windows NTFS configuration, as the maximum [innodb\_page\_size](#_bookmark7) is also 64KB.

For page compression to work on Windows, the file system must be created with a cluster size smaller than 4K, and the [innodb\_page\_size](#_bookmark7) must be at least twice the size of the compression unit. For example, for page compression to work on Windows, you could build the file system with a cluster size of 512 Bytes (which has a compression unit of 8KB) and initialize InnoDB with an [innodb\_page\_size](#_bookmark7) value of 16K or greater.

**Enabling** **Page** **Compression**

To enable page compression, specify the COMPRESSION attribute in the CREATE TABLE statement. For example:

CREATE TABLE t1 (c1 INT) COMPRESSION="zlib";

You can also enable page compression in an ALTER TABLE statement. However, ALTER TABLE ... COMPRESSION only updates the tablespace compression attribute. Writes to the tablespace that occur after setting the new compression algorithm use the new setting, but to apply the new compression algorithm to existing pages, you must rebuild the table using OPTIMIZE TABLE.

ALTER TABLE t1 COMPRESSION="zlib";

OPTIMIZE TABLE t1;

**Disabling** **Page** **Compression**

To disable page compression, set COMPRESSION=None using ALTER TABLE. Writes to the tablespace that occur after setting COMPRESSION=None no longer use page compression. To uncompress existing pages, you must rebuild the table using OPTIMIZE TABLE after setting COMPRESSION=None.

ALTER TABLE t1 COMPRESSION="None";

OPTIMIZE TABLE t1;

**Page** **Compression** **Metadata**

Page compression metadata is found in the Information Schema INNODB\_TABLESPACES table, in the following columns:



**INFORMATION\_SCHEMA.INNODB\_TABLESPACES** **WHERE** **NAME='employees/employees'\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: 45

NAME: employees/employees

FS\_BLOCK\_SIZE: 4096

FILE\_SIZE: 23068672

ALLOCATED\_SIZE: 19415040

**WHERE** **CREATE\_OPTIONS** **LIKE** **'%COMPRESSION=%';**

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

| TABLE\_NAME | TABLE\_SCHEMA | CREATE\_OPTIONS |

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

| employees | test | COMPRESSION="zlib" |

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

• FS\_BLOCK\_SIZE: The file system block size, which is the unit size used for hole punching.

• FILE\_SIZE: The apparent size of the file, which represents the maximum size of the file, uncompressed.

• ALLOCATED\_SIZE: The actual size of the file, which is the amount of space allocated on disk.

**Note**

On Unix-like systems, ls -l *tablespace\_name*.ibd shows the apparent file size (equivalent to FILE\_SIZE) in bytes. To view the actual amount of space allocated on disk (equivalent to ALLOCATED\_SIZE), use du --block-size=1 *tablespace\_name*.ibd. The --block-size=1 option prints the allocated space in bytes instead of blocks, so that it can be compared to ls -l output.

Use SHOW CREATE TABLE to view the current page compression setting (Zlib, Lz4, or None). A table may contain a mix of pages with different compression settings.

In the following example, page compression metadata for the employees table is retrieved from the Information Schema INNODB\_TABLESPACES table.

# Create the employees table with Zlib page compression

CREATE TABLE employees (

emp\_no

birth\_date

first\_name

last\_name

gender

hire\_date DATE NOT NULL,

PRIMARY KEY (emp\_no)

) COMPRESSION="zlib";

# Insert data (not shown)

# Query page compression metadata in INFORMATION\_SCHEMA.INNODB\_TABLESPACES

mysql> **SELECT** **SPACE,** **NAME,** **FS\_BLOCK\_SIZE,** **FILE\_SIZE,** **ALLOCATED\_SIZE** **FROM**

INT

DATE

VARCHAR(14)

VARCHAR(16)

ENUM ('M','F')

NULL,

NULL,

NULL,

NULL,

NULL,

NOT

NOT

NOT

NOT

NOT

Page compression metadata for the employees table shows that the apparent file size is 23068672 bytes while the actual file size (with page compression) is 19415040 bytes. The file system block size is 4096 bytes, which is the block size used for hole punching.

**Identifying** **Tables** **Using** **Page** **Compression**

To identify tables for which page compression is enabled, you can check the Information Schema TABLES table's CREATE\_OPTIONS column for tables defined with the COMPRESSION attribute:

mysql> **SELECT** **TABLE\_NAME,** **TABLE\_SCHEMA,** **CREATE\_OPTIONS** **FROM** **INFORMATION\_SCHEMA.TABLES**

SHOW CREATE TABLE also shows the COMPRESSION attribute, if used.

**Page** **Compression** **Limitations** **and** **Usage** **Notes**

• Page compression is disabled if the file system block size (or compression unit size on Windows) \* 2 > [innodb\_page\_size](#_bookmark7).

• Page compression is not supported for tables that reside in shared tablespaces, which include the system tablespace, temporary tablespaces, and general tablespaces.

• Page compression is not supported for undo log tablespaces.

• Page compression is not supported for redo log pages.

• R-tree pages, which are used for spatial indexes, are not compressed.

• Pages that belong to compressed tables (ROW\_FORMAT=COMPRESSED) are left as-is.

• During recovery, updated pages are written out in an uncompressed form.

• Loading a page-compressed tablespace on a server that does not support the compression algorithm that was used causes an I/O error.

• Before downgrading to an earlier version of MySQL that does not support page compression, uncompress the tables that use the page compression feature. To uncompress a table, run ALTER TABLE ... COMPRESSION=None and OPTIMIZE TABLE.

• Page-compressed tablespaces can be copied between Linux and Windows servers if the compression algorithm that was used is available on both servers.

• Preserving page compression when moving a page-compressed tablespace file from one host to another requires a utility that preserves sparse files.

• Better page compression may be achieved on Fusion-io hardware with NVMFS than on other platforms, as NVMFS is designed to take advantage of punch hole functionality.

• Using the page compression feature with a large InnoDB page size and relatively small file system block size could result in write amplification. For example, a maximum InnoDB page size of 64KB with a 4KB file system block size may improve compression but may also increase demand on the buffer pool, leading to increased I/O and potential write amplification.

**15.10** **InnoDB** **Row** **Formats**

The row format of a table determines how its rows are physically stored, which in turn can affect the performance of queries and DML operations. As more rows fit into a single disk page, queries and index lookups can work faster, less cache memory is required in the buffer pool, and less I/O is required to write out updated values.

The data in each table is divided into pages. The pages that make up each table are arranged in a tree data structure called a B-tree index. Table data and secondary indexes both use this type of structure. The B-tree index that represents an entire table is known as the clustered index, which is organized according to the primary key columns. The nodes of a clustered index data structure contain the values of all columns in the row. The nodes of a secondary index structure contain the values of index columns and primary key columns.

Variable-length columns are an exception to the rule that column values are stored in B-tree index nodes. Variable-length columns that are too long to fit on a B-tree page are stored on separately allocated disk pages called overflow pages. Such columns are referred to as off-page columns. The values of off-page columns are stored in singly-linked lists of overflow pages, with each such column having its own list of one or more overflow pages. Depending on column length, all or a prefix of variable-length column values are stored in the B-tree to avoid wasting storage and having to read a separate page.

The InnoDB storage engine supports four row formats: REDUNDANT, COMPACT, DYNAMIC, and

COMPRESSED.

**Table** **15.15** **InnoDB** **Row** **Format** **Overview**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Row** **Format** | **Compact** **Storage** **Characteristics** | **Enhanced**  **Variable-**  **Length**  **Column**  **Storage** | **Large** **Index**  **Key** **Prefix**  **Support** | **Compression** **Support** | **Supported**  **Tablespace**  **Types** |
| REDUNDANT | No | No | No | No | system, file-per- table, general |
| COMPACT | Yes | No | No | No | system, file-per- table, general |
| DYNAMIC | Yes | Yes | Yes | No | system, file-per- table, general |
| COMPRESSED | Yes | Yes | Yes | Yes | file-per-table,  general |

The topics that follow describe row format storage characteristics and how to define and determine the row format of a table.

• [REDUNDANT Row Format](#_bookmark36)

• [COMPACT Row Format](#_bookmark37)

• [DYNAMIC Row Format](#_bookmark23)

• [COMPRESSED Row Format](#_bookmark38)

• [Defining the Row Format of a Table](#_bookmark39)

• [Determining the Row Format of a Table](#_bookmark40)

**REDUNDANT** **Row** **Format**

The REDUNDANT format provides compatibility with older versions of MySQL.

Tables that use the REDUNDANT row format store the first 768 bytes of variable-length column values (VARCHAR, VARBINARY, and BLOB and TEXT types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short BLOB column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many BLOB columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

**REDUNDANT** **Row** **Format** **Storage** **Characteristics**

The REDUNDANT row format has the following storage characteristics:

• Each index record contains a 6-byte header. The header is used to link together consecutive records, and for row-level locking.

• Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6- byte transaction ID field and a 7-byte roll pointer field.

• If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.

• Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index.

• A record contains a pointer to each field of the record. If the total length of the fields in a record is less than 128 bytes, the pointer is one byte; otherwise, two bytes. The array of pointers is called the

record directory. The area where the pointers point is the data part of the record.

• Internally, fixed-length character columns such as CHAR(10) in stored in fixed-length format. Trailing spaces are not truncated from VARCHAR columns.

• Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

• An SQL NULL value reserves one or two bytes in the record directory. An SQL NULL value reserves zero bytes in the data part of the record if stored in a variable-length column. For a fixed-length column, the fixed length of the column is reserved in the data part of the record. Reserving fixed space for NULL values permits columns to be updated in place from NULL to non-NULL values

without causing index page fragmentation.

**COMPACT** **Row** **Format**

The COMPACT row format reduces row storage space by about 20% compared to the REDUNDANT row format, at the cost of increasing CPU use for some operations. If your workload is a typical one that is limited by cache hit rates and disk speed, COMPACT format is likely to be faster. If the workload is limited by CPU speed, compact format might be slower.

Tables that use the COMPACT row format store the first 768 bytes of variable-length column values (VARCHAR, VARBINARY, and BLOB and TEXT types) in the index record within the B-tree node, with the remainder stored on overflow pages. Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length columns, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

If the value of a column is 768 bytes or less, an overflow page is not used, and some savings in I/O may result, since the value is stored entirely in the B-tree node. This works well for relatively short BLOB column values, but may cause B-tree nodes to fill with data rather than key values, reducing their efficiency. Tables with many BLOB columns could cause B-tree nodes to become too full, and contain too few rows, making the entire index less efficient than if rows were shorter or column values were stored off-page.

**COMPACT** **Row** **Format** **Storage** **Characteristics**

The COMPACT row format has the following storage characteristics:

• Each index record contains a 5-byte header that may be preceded by a variable-length header. The header is used to link together consecutive records, and for row-level locking.

• The variable-length part of the record header contains a bit vector for indicating NULL columns. If the number of columns in the index that can be NULL is *N*, the bit vector occupies CEILING(*N*/8) bytes. (For example, if there are anywhere from 9 to 16 columns that can be NULL, the bit vector uses two bytes.) Columns that are NULL do not occupy space other than the bit in this vector. The variable- length part of the header also contains the lengths of variable-length columns. Each length takes one or two bytes, depending on the maximum length of the column. If all columns in the index are NOT NULL and have a fixed length, the record header has no variable-length part.

• For each non-NULL variable-length field, the record header contains the length of the column in one or two bytes. Two bytes are only needed if part of the column is stored externally in overflow pages or the maximum length exceeds 255 bytes and the actual length exceeds 127 bytes. For an externally stored column, the 2-byte length indicates the length of the internally stored part plus the 20-byte pointer to the externally stored part. The internal part is 768 bytes, so the length is 768+20. The 20-byte pointer stores the true length of the column.

• The record header is followed by the data contents of non-NULL columns.

• Records in the clustered index contain fields for all user-defined columns. In addition, there is a 6- byte transaction ID field and a 7-byte roll pointer field.

• If no primary key is defined for a table, each clustered index record also contains a 6-byte row ID field.

• Each secondary index record contains all the primary key columns defined for the clustered index key that are not in the secondary index. If any of the primary key columns are variable length, the record header for each secondary index has a variable-length part to record their lengths, even if the secondary index is defined on fixed-length columns.

• Internally, for nonvariable-length character sets, fixed-length character columns such as CHAR(10) are stored in a fixed-length format.

Trailing spaces are not truncated from VARCHAR columns.

• Internally, for variable-length character sets such as utf8mb3 and utf8mb4, InnoDB attempts to store CHAR(*N*) in *N* bytes by trimming trailing spaces. If the byte length of a CHAR(*N*) column value exceeds *N* bytes, trailing spaces are trimmed to a minimum of the column value byte length. The maximum length of a CHAR(*N*) column is the maximum character byte length × *N*.

A minimum of *N* bytes is reserved for CHAR(*N*). Reserving the minimum space *N* in many cases enables column updates to be done in place without causing index page fragmentation. By comparison, CHAR(*N*) columns occupy the maximum character byte length × *N*when using the REDUNDANT row format.

Fixed-length columns greater than or equal to 768 bytes are encoded as variable-length fields, which can be stored off-page. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

**DYNAMIC** **Row** **Format**

The DYNAMIC row format offers the same storage characteristics as the COMPACT row format but adds enhanced storage capabilities for long variable-length columns and supports large index key prefixes.

When a table is created with ROW\_FORMAT=DYNAMIC, InnoDB can store long variable-length column values (for VARCHAR, VARBINARY, and BLOB and TEXT types) fully off-page, with the clustered index record containing only a 20-byte pointer to the overflow page. Fixed-length fields greater than or equal to 768 bytes are encoded as variable-length fields. For example, a CHAR(255) column can exceed 768 bytes if the maximum byte length of the character set is greater than 3, as it is with utf8mb4.

Whether columns are stored off-page depends on the page size and the total size of the row. When a row is too long, the longest columns are chosen for off-page storage until the clustered index record fits on the B-tree page. TEXT and BLOB columns that are less than or equal to 40 bytes are stored in line.

The DYNAMIC row format maintains the efficiency of storing the entire row in the index node if it fits (as do the COMPACT and REDUNDANT formats), but the DYNAMIC row format avoids the problem of filling B-tree nodes with a large number of data bytes of long columns. The DYNAMIC row format is based on the idea that if a portion of a long data value is stored off-page, it is usually most efficient to store the entire value off-page. With DYNAMIC format, shorter columns are likely to remain in the B-tree node, minimizing the number of overflow pages required for a given row.

The DYNAMIC row format supports index key prefixes up to 3072 bytes.

Tables that use the DYNAMIC row format can be stored in the system tablespace, file-per-table tablespaces, and general tablespaces. To store DYNAMIC tables in the system tablespace, either disable [innodb\_file\_per\_table](#_bookmark17) and use a regular CREATE TABLE or ALTER TABLE statement, or use the TABLESPACE [=] innodb\_system table option with CREATE TABLE or ALTER TABLE. The [innodb\_file\_per\_table](#_bookmark17) variable is not applicable to general tablespaces, nor is it applicable when using the TABLESPACE [=] innodb\_system table option to store DYNAMIC tables in the system tablespace.

**DYNAMIC** **Row** **Format** **Storage** **Characteristics**

The DYNAMIC row format is a variation of the COMPACT row format. For storage characteristics, see [COMPACT Row Format Storage Characteristics](#_bookmark41).

**COMPRESSED** **Row** **Format**

The COMPRESSED row format offers the same storage characteristics and capabilities as the DYNAMIC row format but adds support for table and index data compression.

The COMPRESSED row format uses similar internal details for off-page storage as the DYNAMIC row format, with additional storage and performance considerations from the table and index data being compressed and using smaller page sizes. With the COMPRESSED row format, the KEY\_BLOCK\_SIZE option controls how much column data is stored in the clustered index, and how much is placed on overflow pages. For more information about the COMPRESSED row format, see [Section 15.9, “InnoDB](#_bookmark16) [Table and Page Compression”](#_bookmark16) .

The COMPRESSED row format supports index key prefixes up to 3072 bytes.

Tables that use the COMPRESSED row format can be created in file-per-table tablespaces or general tablespaces. The system tablespace does not support the COMPRESSED row format. To store a COMPRESSED table in a file-per-table tablespace, the [innodb\_file\_per\_table](#_bookmark17) variable must be enabled. The [innodb\_file\_per\_table](#_bookmark17) variable is not applicable to general tablespaces. General tablespaces support all row formats with the caveat that compressed and uncompressed tables cannot coexist in the same general tablespace due to different physical page sizes. For more information, see Section 15.6.3.3, “General Tablespaces” .

**Compressed** **Row** **Format** **Storage** **Characteristics**

The COMPRESSED row format is a variation of the COMPACT row format. For storage characteristics, see [COMPACT Row Format Storage Characteristics](#_bookmark41).

**Defining** **the** **Row** **Format** **of** **a** **Table**

The default row format for InnoDB tables is defined by [innodb\_default\_row\_format](#_bookmark42) variable, which has a default value of DYNAMIC. The default row format is used when the ROW\_FORMAT table option is not defined explicitly or when ROW\_FORMAT=DEFAULT is specified.

The row format of a table can be defined explicitly using the ROW\_FORMAT table option in a CREATE TABLE or ALTER TABLE statement. For example:

CREATE TABLE t1 (c1 INT) ROW\_FORMAT=DYNAMIC;

An explicitly defined ROW\_FORMAT setting overrides the default row format. Specifying ROW\_FORMAT=DEFAULT is equivalent to using the implicit default.

The [innodb\_default\_row\_format](#_bookmark42) variable can be set dynamically:

mysql> **SET** **GLOBAL** **innodb\_default\_row\_format=DYNAMIC;**

Valid [innodb\_default\_row\_format](#_bookmark42) options include DYNAMIC, COMPACT, and REDUNDANT. The COMPRESSED row format, which is not supported for use in the system tablespace, cannot be defined

as the default. It can only be specified explicitly in a CREATE TABLE or ALTER TABLE statement. Attempting to set the [innodb\_default\_row\_format](#_bookmark42) variable to COMPRESSED returns an error:

mysql> **SET** **GLOBAL** **innodb\_default\_row\_format=COMPRESSED;**

ERROR 1231 (42000): Variable 'innodb\_default\_row\_format'

can't be set to the value of 'COMPRESSED'

Newly created tables use the row format defined by the [innodb\_default\_row\_format](#_bookmark42) variable when a ROW\_FORMAT option is not specified explicitly, or when ROW\_FORMAT=DEFAULT is used. For example, the following CREATE TABLE statements use the row format defined by the [innodb\_default\_row\_format](#_bookmark42) variable.

CREATE TABLE t1 (c1 INT);

CREATE TABLE t2 (c1 INT) ROW\_FORMAT=DEFAULT;

When a ROW\_FORMAT option is not specified explicitly, or when ROW\_FORMAT=DEFAULT is used, an operation that rebuilds a table silently changes the row format of the table to the format defined by the [innodb\_default\_row\_format](#_bookmark42) variable.

Table-rebuilding operations include ALTER TABLE operations that use ALGORITHM=COPY or ALGORITHM=INPLACE where table rebuilding is required. See [Section 15.12.1, “Online DDL](#_bookmark43) [Operations”](#_bookmark43) for more information. OPTIMIZE TABLE is also a table-rebuilding operation.

The following example demonstrates a table-rebuilding operation that silently changes the row format of a table created without an explicitly defined row format.

mysql> **SELECT** **@@innodb\_default\_row\_format;**

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

| @@innodb\_default\_row\_format |

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

| dynamic |

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

mysql> **CREATE** **TABLE** **t1** **(c1** **INT);**

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLES** **WHERE** **NAME** **LIKE** **'test/t1'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 54

NAME: test/t1

FLAG: 33

N\_COLS: 4

SPACE: 35

ROW\_FORMAT: Dynamic

ZIP\_PAGE\_SIZE: 0

SPACE\_TYPE: Single

mysql> **SET** **GLOBAL** **innodb\_default\_row\_format=COMPACT;**

mysql> **ALTER** **TABLE** **t1** **ADD** **COLUMN** **(c2** **INT);**

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLES** **WHERE** **NAME** **LIKE** **'test/t1'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 55

NAME: test/t1

FLAG: 1

N\_COLS: 5

SPACE: 36

ROW\_FORMAT: Compact

ZIP\_PAGE\_SIZE: 0

SPACE\_TYPE: Single

Consider the following potential issues before changing the row format of existing tables from REDUNDANT or COMPACT to DYNAMIC.

• The REDUNDANT and COMPACT row formats support a maximum index key prefix length of 767 bytes whereas DYNAMIC and COMPRESSED row formats support an index key prefix length of 3072 bytes.

In a replication environment, if the [innodb\_default\_row\_format](#_bookmark42) variable is set to DYNAMIC on the source, and set to COMPACT on the replica, the following DDL statement, which does not explicitly define a row format, succeeds on the source but fails on the replica:

CREATE TABLE t1 (c1 INT PRIMARY KEY, c2 VARCHAR(5000), KEY i1(c2(3070)));

For related information, see Section 15.22, “InnoDB Limits” .

• Importing a table that does not explicitly define a row format results in a schema mismatch error if the [innodb\_default\_row\_format](#_bookmark42) setting on the source server differs from the setting on the destination server. For more information, see Section 15.6.1.3, “Importing InnoDB Tables” .

**Determining** **the** **Row** **Format** **of** **a** **Table**

To determine the row format of a table, use SHOW TABLE STATUS:

mysql> **SHOW** **TABLE** **STATUS** **IN** **test1\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Name: t1

Engine: InnoDB

Version: 10

Row\_format: Dynamic

Rows: 0

Avg\_row\_length: 0

Data\_length: 16384

Max\_data\_length: 0

Index\_length: 16384

Data\_free: 0

Auto\_increment: 1

Create\_time: 2016-09-14 16:29:38

Update\_time: NULL

Check\_time: NULL

Collation: utf8mb4\_0900\_ai\_ci

Checksum: NULL

Create\_options:

Comment:

Alternatively, query the Information Schema INNODB\_TABLES table:

mysql> **SELECT** **NAME,** **ROW\_FORMAT** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLES** **WHERE** **NAME='test1/t1';**

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

| NAME | ROW\_FORMAT |

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

| test1/t1 | Dynamic |

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

**15.11** **InnoDB** **Disk** **I/O** **and** **File** **Space** **Management**

As a DBA, you must manage disk I/O to keep the I/O subsystem from becoming saturated, and manage disk space to avoid filling up storage devices. The ACID design model requires a certain amount of I/O that might seem redundant, but helps to ensure data reliability. Within these constraints, InnoDB tries to optimize the database work and the organization of disk files to minimize the amount of disk I/O. Sometimes, I/O is postponed until the database is not busy, or until everything needs to be brought to a consistent state, such as during a database restart after a fast shutdown.

This section discusses the main considerations for I/O and disk space with the default kind of MySQL tables (also known as InnoDB tables):

• Controlling the amount of background I/O used to improve query performance.

• Enabling or disabling features that provide extra durability at the expense of additional I/O.

• Organizing tables into many small files, a few larger files, or a combination of both.

• Balancing the size of redo log files against the I/O activity that occurs when the log files become full.

• How to reorganize a table for optimal query performance.

**15.11.1** **InnoDB** **Disk** **I/O**

InnoDB uses asynchronous disk I/O where possible, by creating a number of threads to handle I/O operations, while permitting other database operations to proceed while the I/O is still in progress. On Linux and Windows platforms, InnoDB uses the available OS and library functions to perform “native” asynchronous I/O. On other platforms, InnoDB still uses I/O threads, but the threads may actually wait for I/O requests to complete; this technique is known as “simulated” asynchronous I/O.

**Read-Ahead**

If InnoDB can determine there is a high probability that data might be needed soon, it performs read- ahead operations to bring that data into the buffer pool so that it is available in memory. Making a few large read requests for contiguous data can be more efficient than making several small, spread-out requests. There are two read-ahead heuristics in InnoDB:

• In sequential read-ahead, if InnoDB notices that the access pattern to a segment in the tablespace is sequential, it posts in advance a batch of reads of database pages to the I/O system.

• In random read-ahead, if InnoDB notices that some area in a tablespace seems to be in the process of being fully read into the buffer pool, it posts the remaining reads to the I/O system.

For information about configuring read-ahead heuristics, see Section 15.8.3.4, “Configuring InnoDB Buffer Pool Prefetching (Read-Ahead)” .

**Doublewrite** **Buffer**

InnoDB uses a novel file flush technique involving a structure called the doublewrite buffer, which is enabled by default in most cases ([innodb\_doublewrite=ON](#_bookmark44)). It adds safety to recovery following an unexpected exit or power outage, and improves performance on most varieties of Unix by reducing the need for fsync() operations.

Before writing pages to a data file, InnoDB first writes them to a storage area called the doublewrite buffer. Only after the write and the flush to the doublewrite buffer has completed does InnoDB write the pages to their proper positions in the data file. If there is an operating system, storage subsystem, or unexpected mysqld process exit in the middle of a page write (causing a torn page condition), InnoDB can later find a good copy of the page from the doublewrite buffer during recovery.

For more information about the doublewrite buffer, see Section 15.6.4, “Doublewrite Buffer” .

**15.11.2** **File** **Space** **Management**

The data files that you define in the configuration file using the [innodb\_data\_file\_path](#_bookmark45) configuration option form the InnoDB system tablespace. The files are logically concatenated to form the system tablespace. There is no striping in use. You cannot define where within the system tablespace your tables are allocated. In a newly created system tablespace, InnoDB allocates space starting from the first data file.

To avoid the issues that come with storing all tables and indexes inside the system tablespace, you can enable the [innodb\_file\_per\_table](#_bookmark17) configuration option (the default), which stores each newly created table in a separate tablespace file (with extension .ibd). For tables stored this way, there is less fragmentation within the disk file, and when the table is truncated, the space is returned to the operating system rather than still being reserved by InnoDB within the system tablespace. For more information, see Section 15.6.3.2, “File-Per-Table Tablespaces” .

You can also store tables in general tablespaces. General tablespaces are shared tablespaces created using CREATE TABLESPACE syntax. They can be created outside of the MySQL data directory, are capable of holding multiple tables, and support tables of all row formats. For more information, see Section 15.6.3.3, “General Tablespaces” .

**Pages,** **Extents,** **Segments,** **and** **Tablespaces**

Each tablespace consists of database pages. Every tablespace in a MySQL instance has the same page size. By default, all tablespaces have a page size of 16KB; you can reduce the page size to 8KB or 4KB by specifying the [innodb\_page\_size](#_bookmark7) option when you create the MySQL instance. You can also increase the page size to 32KB or 64KB. For more information, refer to the [innodb\_page\_size](#_bookmark7) documentation.

The pages are grouped into extents of size 1MB for pages up to 16KB in size (64 consecutive 16KB pages, or 128 8KB pages, or 256 4KB pages). For a page size of 32KB, extent size is 2MB. For page size of 64KB, extent size is 4MB. The “files” inside a tablespace are called segments in InnoDB. (These segments are different from the rollback segment, which actually contains many tablespace segments.)

When a segment grows inside the tablespace, InnoDB allocates the first 32 pages to it one at a time. After that, InnoDB starts to allocate whole extents to the segment. InnoDB can add up to 4 extents at a time to a large segment to ensure good sequentiality of data.

Two segments are allocated for each index in InnoDB. One is for nonleaf nodes of the B-tree, the other is for the leaf nodes. Keeping the leaf nodes contiguous on disk enables better sequential I/O operations, because these leaf nodes contain the actual table data.

Some pages in the tablespace contain bitmaps of other pages, and therefore a few extents in an InnoDB tablespace cannot be allocated to segments as a whole, but only as individual pages.

When you ask for available free space in the tablespace by issuing a SHOW TABLE STATUS statement, InnoDB reports the extents that are definitely free in the tablespace. InnoDB always reserves some extents for cleanup and other internal purposes; these reserved extents are not included in the free space.

When you delete data from a table, InnoDB contracts the corresponding B-tree indexes. Whether the freed space becomes available for other users depends on whether the pattern of deletes frees individual pages or extents to the tablespace. Dropping a table or deleting all rows from it is guaranteed to release the space to other users, but remember that deleted rows are physically removed only by the purge operation, which happens automatically some time after they are no longer needed for transaction rollbacks or consistent reads. (See Section 15.3, “InnoDB Multi-Versioning” .)

**Configuring** **the** **Percentage** **of** **Reserved** **File** **Segment** **Pages**

The [innodb\_segment\_reserve\_factor](#_bookmark47) variable, introduced in MySQL 8.0.26, is an advanced feature that permits defining the percentage of tablespace file segment pages reserved as empty pages. A percentage of pages are reserved for future growth so that pages in the B-tree can be allocated contiguously. The ability to modify the percentage of reserved pages permits fine-tuning InnoDB to address issues of data fragmentation or inefficient use of storage space.

The setting is applicable to file-per-table and general tablespaces. The [innodb\_segment\_reserve\_factor](#_bookmark47) default setting is 12.5 percent, which is the same percentage of pages reserved in previous MySQL releases.

The [innodb\_segment\_reserve\_factor](#_bookmark47) variable is dynamic and can be configured using a SET statement. For example:

mysql> SET GLOBAL innodb\_segment\_reserve\_factor=10;

**How** **Pages** **Relate** **to** **Table** **Rows**

For for 4KB, 8KB, 16KB, and 32KB [innodb\_page\_size](#_bookmark7) settings, the maximum row length is slightly less than half a database page size. For example, the maximum row length is slightly less than 8KB for the default 16KB InnoDB page size. For a 64KB [innodb\_page\_size](#_bookmark7) setting, the maximum row length is slightly less than 16KB.

If a row does not exceed the maximum row length, all of it is stored locally within the page. If a row exceeds the maximum row length, variable-length columns are chosen for external off-page storage until the row fits within the maximum row length limit. External off-page storage for variable-length columns differs by row format:

• *COMPACT* *and* *REDUNDANT* *Row* *Formats*

When a variable-length column is chosen for external off-page storage, InnoDB stores the first 768 bytes locally in the row, and the rest externally into overflow pages. Each such column has its own list of overflow pages. The 768-byte prefix is accompanied by a 20-byte value that stores the true length of the column and points into the overflow list where the rest of the value is stored. See [Section 15.10, “InnoDB Row Formats”](#_bookmark35) .

• *DYNAMIC* *and* *COMPRESSED* *Row* *Formats*

When a variable-length column is chosen for external off-page storage, InnoDB stores a 20-byte pointer locally in the row, and the rest externally into overflow pages. See [Section 15.10, “InnoDB](#_bookmark35) [Row Formats”](#_bookmark35) .

LONGBLOB and LONGTEXT columns must be less than 4GB, and the total row length, including BLOB and TEXT columns, must be less than 4GB.

**15.11.3** **InnoDB** **Checkpoints**

Making your log files very large may reduce disk I/O during checkpointing. It often makes sense to set the total size of the log files as large as the buffer pool or even larger.

**How** **Checkpoint** **Processing** **Works**

InnoDB implements a checkpoint mechanism known as fuzzy checkpointing. InnoDB flushes modified database pages from the buffer pool in small batches. There is no need to flush the buffer pool in one single batch, which would disrupt processing of user SQL statements during the checkpointing process.

During crash recovery, InnoDB looks for a checkpoint label written to the log files. It knows that all modifications to the database before the label are present in the disk image of the database. Then InnoDB scans the log files forward from the checkpoint, applying the logged modifications to the database.

**15.11.4** **Defragmenting** **a** **Table**

Random insertions into or deletions from a secondary index can cause the index to become fragmented. Fragmentation means that the physical ordering of the index pages on the disk is not close to the index ordering of the records on the pages, or that there are many unused pages in the 64-page blocks that were allocated to the index.

One symptom of fragmentation is that a table takes more space than it “should” take. How much that is exactly, is difficult to determine. All InnoDB data and indexes are stored in B-trees, and their fill factor may vary from 50% to 100%. Another symptom of fragmentation is that a table scan such as this takes more time than it “should” take:

SELECT COUNT(\*) FROM t WHERE *non\_indexed\_column* <> 12345;

The preceding query requires MySQL to perform a full table scan, the slowest type of query for a large table.

To speed up index scans, you can periodically perform a “null” ALTER TABLE operation, which causes MySQL to rebuild the table:

ALTER TABLE *tbl\_name* ENGINE=INNODB

You can also use ALTER TABLE *tbl\_name* FORCE to perform a “null” alter operation that rebuilds the table.



Both ALTER TABLE *tbl\_name* ENGINE=INNODB and ALTER TABLE *tbl\_name* FORCE use [online](#_bookmark48) [DDL](#_bookmark48). For more information, see [Section 15.12, “InnoDB Online DDL”](#_bookmark48)and.

Another way to perform a defragmentation operation is to use mysqldump to dump the table to a text file, drop the table, and reload it from the dump file.

If the insertions into an index are always ascending and records are deleted only from the end, the InnoDB filespace management algorithm guarantees that fragmentation in the index does not occur.

**15.11.5** **Reclaiming** **Disk** **Space** **with** **TRUNCATE** **TABLE**

To reclaim operating system disk space when truncating an InnoDB table, the table must be stored in its own .ibd file. For a table to be stored in its own .ibd file, [innodb\_file\_per\_table](#_bookmark17) must enabled when the table is created. Additionally, there cannot be a foreign key constraint between the table being truncated and other tables, otherwise the TRUNCATE TABLE operation fails. A foreign key constraint between two columns in the same table, however, is permitted.

When a table is truncated, it is dropped and re-created in a new .ibd file, and the freed space is returned to the operating system. This is in contrast to truncating InnoDB tables that are stored within the InnoDB system tablespace (tables created when innodb\_file\_per\_table=OFF) and tables stored in shared general tablespaces, where only InnoDB can use the freed space after the table is truncated.

The ability to truncate tables and return disk space to the operating system also means that physical backups can be smaller. Truncating tables that are stored in the system tablespace (tables created when innodb\_file\_per\_table=OFF) or in a general tablespace leaves blocks of unused space in the tablespace.

**15.12** **InnoDB** **and** **Online** **DDL**

The online DDL feature provides support for instant and in-place table alterations and concurrent DML. Benefits of this feature include:

• Improved responsiveness and availability in busy production environments, where making a table unavailable for minutes or hours is not practical.

• For in-place operations, the ability to adjust the balance between performance and concurrency during DDL operations using the LOCK clause. See [The LOCK clause](#_bookmark49).

• Less disk space usage and I/O overhead than the table-copy method.

**Note**

ALGORITHM=INSTANT support is available for ADD COLUMN and other operations in MySQL 8.0.12.

Typically, you do not need to do anything special to enable online DDL. By default, MySQL performs the operation instantly or in place, as permitted, with as little locking as possible.

You can control aspects of a DDL operation using the ALGORITHM and LOCK clauses of the ALTER TABLE statement. These clauses are placed at the end of the statement, separated from the table and column specifications by commas. For example:

ALTER TABLE *tbl\_name* ADD PRIMARY KEY (*column*), ALGORITHM=INPLACE, LOCK=NONE;

The LOCK clause may be used for operations that are performed in place and is useful for fine-tuning the degree of concurrent access to the table during operations. Only LOCK=DEFAULT is supported for operations that are performed instantly. The ALGORITHM clause is primarily intended for performance comparisons and as a fallback to the older table-copying behavior in case you encounter any issues. For example:

• To avoid accidentally making the table unavailable for reads, writes, or both, during an in-place ALTER TABLE operation, specify a clause on the ALTER TABLE statement such as LOCK=NONE (permit reads and writes) or LOCK=SHARED (permit reads). The operation halts immediately if the requested level of concurrency is not available.

• To compare performance between algorithms, run a statement with ALGORITHM=INSTANT, ALGORITHM=INPLACE and ALGORITHM=COPY. You can also run a statement with the old\_alter\_table configuration option enabled to force the use of ALGORITHM=COPY.

• To avoid tying up the server with an ALTER TABLE operation that copies the table, include ALGORITHM=INSTANT or ALGORITHM=INPLACE. The statement halts immediately if it cannot use the specified algorithm.

**15.12.1** **Online** **DDL** **Operations**

Online support details, syntax examples, and usage notes for DDL operations are provided under the following topics in this section.

• [Index Operations](#_bookmark50)

• [Primary Key Operations](#_bookmark51)

• [Column Operations](#_bookmark52)

• [Generated Column Operations](#_bookmark53)

• [Foreign Key Operations](#_bookmark54)

• [Table Operations](#_bookmark55)

• [Tablespace Operations](#_bookmark56)

• [Partitioning Operations](#_bookmark57)

**Index** **Operations**

The following table provides an overview of online DDL support for index operations. An asterisk indicates additional information, an exception, or a dependency. For details, see [Syntax and Usage](#_bookmark58) [Notes](#_bookmark58).

**Table** **15.16** **Online** **DDL** **Support** **for** **Index** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Creating or  adding a  secondary  index | No | Yes | No | Yes | No |
| Dropping an  index | No | Yes | No | Yes | Yes |
| Renaming an index | No | Yes | No | Yes | Yes |
| Adding a  FULLTEXT  index | No | Yes\* | No\* | No | No |
| Adding a SPATIAL index | No | Yes | No | No | No |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Changing the index type | Yes | Yes | No | Yes | Yes |

**Syntax** **and** **Usage** **Notes**

• Creating or adding a secondary index CREATE INDEX *name* ON *table* (*col\_list*); ALTER TABLE *tbl\_name* ADD INDEX *name* (*col\_list*);

The table remains available for read and write operations while the index is being created. The CREATE INDEX statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

Online DDL support for adding secondary indexes means that you can generally speed the overall process of creating and loading a table and associated indexes by creating the table without secondary indexes, then adding secondary indexes after the data is loaded.

A newly created secondary index contains only the committed data in the table at the time the CREATE INDEX or ALTER TABLE statement finishes executing. It does not contain any uncommitted values, old versions of values, or values marked for deletion but not yet removed from the old index.

Some factors affect the performance, space usage, and semantics of this operation. For details, see [Section 15.12.8, “Online DDL Limitations”](#_bookmark59) .

• Dropping an index DROP INDEX *name* ON *table*; ALTER TABLE *tbl\_name* DROP INDEX *name*;

The table remains available for read and write operations while the index is being dropped. The DROP INDEX statement only finishes after all transactions that are accessing the table are completed, so that the initial state of the index reflects the most recent contents of the table.

• Renaming an index ALTER TABLE *tbl\_name* RENAME INDEX *old\_index\_name* TO *new\_index\_name*, ALGORITHM=INPLACE, LOCK=NONE;

• Adding a FULLTEXT index CREATE FULLTEXT INDEX *name* ON table(*column*);

Adding the first FULLTEXT index rebuilds the table if there is no user-defined FTS\_DOC\_ID column. Additional FULLTEXT indexes may be added without rebuilding the table.

• Adding a SPATIAL index

CREATE TABLE geom (g GEOMETRY NOT NULL);

ALTER TABLE geom ADD SPATIAL INDEX(g), ALGORITHM=INPLACE, LOCK=SHARED;

• Changing the index type (USING {BTREE | HASH}) ALTER TABLE *tbl\_name* DROP INDEX i1, ADD INDEX i1(*key\_part,...*) USING BTREE, ALGORITHM=INSTANT;

**Primary** **Key** **Operations**

The following table provides an overview of online DDL support for primary key operations. An asterisk indicates additional information, an exception, or a dependency. See [Syntax and Usage Notes](#_bookmark60).

**Table** **15.17** **Online** **DDL** **Support** **for** **Primary** **Key** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Adding a  primary key | No | Yes\* | Yes\* | Yes | No |
| Dropping a  primary key | No | No | Yes | No | No |
| Dropping a primary key and adding another | No | Yes | Yes | Yes | No |

**Syntax** **and** **Usage** **Notes**

• Adding a primary key

ALTER TABLE *tbl\_name* ADD PRIMARY KEY (*column*), ALGORITHM=INPLACE, LOCK=NONE;

Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation. ALGORITHM=INPLACE is not permitted under certain conditions if columns have to be converted to

NOT NULL.

Restructuring the clustered index always requires copying of table data. Thus, it is best to define the primary key when you create a table, rather than issuing ALTER TABLE ... ADD PRIMARY KEY later.

When you create a UNIQUE or PRIMARY KEY index, MySQL must do some extra work. For UNIQUE indexes, MySQL checks that the table contains no duplicate values for the key. For a PRIMARY KEY index, MySQL also checks that none of the PRIMARY KEY columns contains a NULL.

When you add a primary key using the ALGORITHM=COPY clause, MySQL converts NULL values in the associated columns to default values: 0 for numbers, an empty string for character-based columns and BLOBs, and 0000-00-00 00:00:00 for DATETIME. This is a non-standard behavior that Oracle recommends you not rely on. Adding a primary key using ALGORITHM=INPLACE is only permitted when the SQL\_MODE setting includes the strict\_trans\_tables or strict\_all\_tables flags; when the SQL\_MODE setting is strict, ALGORITHM=INPLACE is permitted, but the statement can still fail if the requested primary key columns contain NULL values. The ALGORITHM=INPLACE behavior is more standard-compliant.

If you create a table without a primary key, InnoDB chooses one for you, which can be the first UNIQUE key defined on NOT NULL columns, or a system-generated key. To avoid uncertainty and the potential space requirement for an extra hidden column, specify the PRIMARY KEY clause as part of the CREATE TABLE statement.

MySQL creates a new clustered index by copying the existing data from the original table to a temporary table that has the desired index structure. Once the data is completely copied to the temporary table, the original table is renamed with a different temporary table name. The temporary table comprising the new clustered index is renamed with the name of the original table, and the original table is dropped from the database.

The online performance enhancements that apply to operations on secondary indexes do not apply to the primary key index. The rows of an InnoDB table are stored in a clustered index organized based on the primary key, forming what some database systems call an “index-organized table” . Because the table structure is closely tied to the primary key, redefining the primary key still requires copying the data.

When an operation on the primary key uses ALGORITHM=INPLACE, even though the data is still copied, it is more efficient than using ALGORITHM=COPY because:

• No undo logging or associated redo logging is required for ALGORITHM=INPLACE. These operations add overhead to DDL statements that use ALGORITHM=COPY.

• The secondary index entries are pre-sorted, and so can be loaded in order.

• The change buffer is not used, because there are no random-access inserts into the secondary indexes.

• Dropping a primary key ALTER TABLE *tbl\_name* DROP PRIMARY KEY, ALGORITHM=COPY;

Only ALGORITHM=COPY supports dropping a primary key without adding a new one in the same ALTER TABLE statement.

• Dropping a primary key and adding another ALTER TABLE *tbl\_name* DROP PRIMARY KEY, ADD PRIMARY KEY (*column*), ALGORITHM=INPLACE, LOCK=NONE; Data is reorganized substantially, making it an expensive operation.

**Column** **Operations**

The following table provides an overview of online DDL support for column operations. An asterisk indicates additional information, an exception, or a dependency. For details, see [Syntax and Usage](#_bookmark61) [Notes](#_bookmark61).

**Table** **15.18** **Online** **DDL** **Support** **for** **Column** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Adding a  column | Yes\* | Yes | No\* | Yes\* | Yes |
| Dropping a  column | Yes\* | Yes | Yes | Yes | Yes |
| Renaming a  column | Yes\* | Yes | No | Yes\* | Yes |
| Reordering  columns | No | Yes | Yes | Yes | No |
| Setting a column default value | Yes | Yes | No | Yes | Yes |
| Changing the column data type | No | No | Yes | No | No |
| Extending  VARCHAR  column size | No | Yes | No | Yes | Yes |
| Dropping the column default value | Yes | Yes | No | Yes | Yes |
| Changing the auto-increment value | No | Yes | No | Yes | No\* |

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Making a column NULL | No | Yes | Yes\* | Yes | No |
| Making a  column NOT  NULL | No | Yes\* | Yes\* | Yes | No |
| Modifying the definition of an ENUM or SET column | Yes | Yes | No | Yes | Yes |

**Syntax** **and** **Usage** **Notes**

• Adding a column

ALTER TABLE *tbl\_name* ADD COLUMN *column\_name* *column\_definition*, ALGORITHM=INSTANT;

INSTANT is the default algorithm as of MySQL 8.0.12, and INPLACE before that.

The following limitations apply when the INSTANT algorithm adds a column:

• A statement cannot combine the addition of a column with other ALTER TABLE actions that do not support the INSTANT algorithm.

• The INSTANT algorithm can add a column at any position in the table. Before MySQL 8.0.29, the INSTANT algorithm could only add a column as the last column of the table.

• Columns cannot be added to tables that use ROW\_FORMAT=COMPRESSED, tables with a FULLTEXT index, tables that reside in the data dictionary tablespace, or temporary tables. Temporary tables only support ALGORITHM=COPY.

• MySQL checks the row size when the INSTANT algorithm adds a column, and throws the following error if the addition exceeds the limit.

ERROR 4092 (HY000): Column can't be added with ALGORITHM=INSTANT as

after this max possible row size crosses max permissible row size. Try

ALGORITHM=INPLACE/COPY.

Before MySQL 8.0.29, MySQL does not check the row size when the INSTANT algorithm adds a column. However, MySQL does check the row size during DML operations that insert and update rows in the table.

• The maximum number of columns in the internal representation of the table cannot exceed 1022 after column addition with the INSTANT algorithm. The error message is:

ERROR 4158 (HY000): Column can't be added to *tbl\_name* with

ALGORITHM=INSTANT anymore. Please try ALGORITHM=INPLACE/COPY Multiple columns may be added in the same ALTER TABLE statement. For example:

ALTER TABLE t1 ADD COLUMN c2 INT, ADD COLUMN c3 INT, ALGORITHM=INSTANT;

A new row version is created after each ALTER TABLE ... ALGORITHM=INSTANT operation that adds one or more columns, drops one or more columns, or adds and drops one or more columns in the same operation. The INFORMATION\_SCHEMA.INNODB\_TABLES.TOTAL\_ROW\_VERSIONS column tracks the number of row versions for a table. The value is incremented each time a column is instantly added or dropped. The initial value is 0.

mysql> SELECT NAME, TOTAL\_ROW\_VERSIONS FROM INFORMATION\_SCHEMA.INNODB\_TABLES

WHERE NAME LIKE 'test/t1';

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

| NAME | TOTAL\_ROW\_VERSIONS |

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

|  |  |
| --- | --- |
| | test/t1 | | 0 | |

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

When a table with instantly added or dropped columns is rebuilt by table-rebuilding ALTER TABLE or OPTIMIZE TABLE operation, the TOTAL\_ROW\_VERSIONS value is reset to 0. The maximum number of row versions permitted is 64, as each row version requires additional space for table metadata. When the row version limit is reached, ADD COLUMN and DROP COLUMN operations using ALGORITHM=INSTANT are rejected with an error message that recommends rebuilding the table using the COPY or INPLACE algorithm.

ERROR 4080 (HY000): Maximum row versions reached for table test/t1. No

more columns can be added or dropped instantly. Please use COPY/INPLACE.

The following INFORMATION\_SCHEMA columns provide additional metadata for instantly added columns. Refer to the descriptions of those columns for more information. See Section 26.4.9, “The INFORMATION\_SCHEMA INNODB\_COLUMNS Table” , and Section 26.4.23, “The INFORMATION\_SCHEMA INNODB\_TABLES Table” .

• INNODB\_COLUMNS.DEFAULT\_VALUE

• INNODB\_COLUMNS.HAS\_DEFAULT

• INNODB\_TABLES.INSTANT\_COLS

Concurrent DML is not permitted when adding an auto-increment column. Data is reorganized substantially, making it an expensive operation. At a minimum, ALGORITHM=INPLACE, LOCK=SHARED is required.

The table is rebuilt if ALGORITHM=INPLACE is used to add a column.

• Dropping a column

ALTER TABLE *tbl\_name* DROP COLUMN *column\_name*, ALGORITHM=INSTANT;

INSTANT is the default algorithm as of MySQL 8.0.29, and INPLACE before that.

The following limitations apply when the INSTANT algorithm is used to drop a column:

• Dropping a column cannot be combined in the same statement with other ALTER TABLE actions that do not support ALGORITHM=INSTANT.

• Columns cannot be dropped from tables that use ROW\_FORMAT=COMPRESSED, tables with a FULLTEXT index, tables that reside in the data dictionary tablespace, or temporary tables. Temporary tables only support ALGORITHM=COPY.

Multiple columns may be dropped in the same ALTER TABLE statement; for example:

ALTER TABLE t1 DROP COLUMN c4, DROP COLUMN c5, ALGORITHM=INSTANT;

Each time a column is added or dropped using ALGORITHM=INSTANT, a new row version is created. The INFORMATION\_SCHEMA.INNODB\_TABLES.TOTAL\_ROW\_VERSIONS column tracks the number of row versions for a table. The value is incremented each time a column is instantly added or dropped. The initial value is 0.

mysql> SELECT NAME, TOTAL\_ROW\_VERSIONS FROM INFORMATION\_SCHEMA.INNODB\_TABLES

WHERE NAME LIKE 'test/t1';

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

| NAME | TOTAL\_ROW\_VERSIONS |

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

| test/t1 | 0 |

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

When a table with instantly added or dropped columns is rebuilt by table-rebuilding ALTER TABLE or OPTIMIZE TABLE operation, the TOTAL\_ROW\_VERSIONS value is reset to 0. The maximum number of row versions permitted is 64, as each row version requires additional space for table metadata. When the row version limit is reached, ADD COLUMN and DROP COLUMN operations using ALGORITHM=INSTANT are rejected with an error message that recommends rebuilding the table using the COPY or INPLACE algorithm.

ERROR 4080 (HY000): Maximum row versions reached for table test/t1. No

more columns can be added or dropped instantly. Please use COPY/INPLACE.

If an algorithm other than ALGORITHM=INSTANT is used, data is reorganized substantially, making it an expensive operation.

• Renaming a column ALTER TABLE *tbl* CHANGE *old\_col\_name* *new\_col\_name* *data\_type*, ALGORITHM=INSTANT, LOCK=NONE;

ALGORITHM=INSTANT support for renaming a column was added in MySQL 8.0.28. Earlier MySQL Server releases support only ALGORITHM=INPLACE and ALGORITHM=COPY when renaming a column.

To permit concurrent DML, keep the same data type and only change the column name.

When you keep the same data type and [NOT] NULL attribute, only changing the column name, the operation can always be performed online.

Renaming a column referenced from another table is only permitted with ALGORITHM=INPLACE. If you use ALGORITHM=INSTANT, ALGORITHM=COPY, or some other condition that causes the operation to use those algorithms, the ALTER TABLE statement fails.

ALGORITHM=INSTANT supports renaming a virtual column; ALGORITHM=INPLACE does not.

ALGORITHM=INSTANT and ALGORITHM=INPLACE do not support renaming a column when adding or dropping a virtual column in the same statement. In this case, only ALGORITHM=COPY is supported.

• Reordering columns

To reorder columns, use FIRST or AFTER in CHANGE or MODIFY operations. ALTER TABLE *tbl\_name* MODIFY COLUMN *col\_name* *column\_definition* FIRST, ALGORITHM=INPLACE, LOCK=NONE; Data is reorganized substantially, making it an expensive operation.

• Changing the column data type ALTER TABLE *tbl\_name* CHANGE c1 c1 BIGINT, ALGORITHM=COPY; Changing the column data type is only supported with ALGORITHM=COPY.

• Extending VARCHAR column size ALTER TABLE *tbl\_name* CHANGE COLUMN c1 c1 VARCHAR(255), ALGORITHM=INPLACE, LOCK=NONE;

The number of length bytes required by a VARCHAR column must remain the same. For VARCHAR columns of 0 to 255 bytes in size, one length byte is required to encode the value. For VARCHAR columns of 256 bytes in size or more, two length bytes are required. As a result, in-place ALTER TABLE only supports increasing VARCHAR column size from 0 to 255 bytes, or from 256 bytes to a greater size. In-place ALTER TABLE does not support increasing the size of a VARCHAR column from



less than 256 bytes to a size equal to or greater than 256 bytes. In this case, the number of required length bytes changes from 1 to 2, which is only supported by a table copy (ALGORITHM=COPY). For example, attempting to change VARCHAR column size for a single byte character set from VARCHAR(255) to VARCHAR(256) using in-place ALTER TABLE returns this error:

|  |  |
| --- | --- |
| ALTER TABLE *tbl\_name* ALGORITHM=INPLACE, CHANGE COLUMN c1  ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason:  column type INPLACE. Try ALGORITHM=COPY. | c1 VARCHAR(256);  Cannot change |

**Note**

The byte length of a VARCHAR column is dependant on the byte length of the character set.

Decreasing VARCHAR size using in-place ALTER TABLE is not supported. Decreasing VARCHAR size requires a table copy (ALGORITHM=COPY).

• Setting a column default value ALTER TABLE *tbl\_name* ALTER COLUMN *col* SET DEFAULT *literal*, ALGORITHM=INSTANT; Only modifies table metadata. Default column values are stored in the data dictionary.

• Dropping a column default value ALTER TABLE *tbl* ALTER COLUMN *col* DROP DEFAULT, ALGORITHM=INSTANT;

• Changing the auto-increment value ALTER TABLE *table* AUTO\_INCREMENT=*next\_value*, ALGORITHM=INPLACE, LOCK=NONE; Modifies a value stored in memory, not the data file.

In a distributed system using replication or sharding, you sometimes reset the auto-increment counter for a table to a specific value. The next row inserted into the table uses the specified value for its auto-increment column. You might also use this technique in a data warehousing environment where you periodically empty all the tables and reload them, and restart the auto- increment sequence from 1.

• Making a column NULL ALTER TABLE tbl\_name MODIFY COLUMN *column\_name* *data\_type* NULL, ALGORITHM=INPLACE, LOCK=NONE; Rebuilds the table in place. Data is reorganized substantially, making it an expensive operation.

• Making a column NOT NULL ALTER TABLE *tbl\_name* MODIFY COLUMN *column\_name* *data\_type* NOT NULL, ALGORITHM=INPLACE, LOCK=NONE;

Rebuilds the table in place. STRICT\_ALL\_TABLES or STRICT\_TRANS\_TABLES SQL\_MODE is required for the operation to succeed. The operation fails if the column contains NULL values. The server prohibits changes to foreign key columns that have the potential to cause loss of referential integrity. See Section 13.1.9, “ALTER TABLE Statement” . Data is reorganized substantially, making it an expensive operation.

• Modifying the definition of an ENUM or SET column

CREATE TABLE t1 (c1 ENUM('a', 'b', 'c'));

ALTER TABLE t1 MODIFY COLUMN c1 ENUM('a', 'b', 'c', 'd'), ALGORITHM=INSTANT;

Modifying the definition of an ENUM or SET column by adding new enumeration or set members to the *end* of the list of valid member values may be performed instantly or in place, as long as the storage size of the data type does not change. For example, adding a member to a SET column that has 8 members changes the required storage per value from 1 byte to 2 bytes; this requires a table copy.

Adding members in the middle of the list causes renumbering of existing members, which requires a table copy.

**Generated** **Column** **Operations**

The following table provides an overview of online DDL support for generated column operations. For details, see [Syntax and Usage Notes](#_bookmark62).

**Table** **15.19** **Online** **DDL** **Support** **for** **Generated** **Column** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Adding a STORED column | No | No | Yes | No | No |
| Modifying STORED column order | No | No | Yes | No | No |
| Dropping a STORED column | No | Yes | Yes | Yes | No |
| Adding a  VIRTUAL  column | Yes | Yes | No | Yes | Yes |
| Modifying  VIRTUAL  column order | No | No | Yes | No | No |
| Dropping a  VIRTUAL  column | Yes | Yes | No | Yes | Yes |

**Syntax** **and** **Usage** **Notes**

• Adding a STORED column ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) STORED), ALGORITHM=COPY;

ADD COLUMN is not an in-place operation for stored columns (done without using a temporary table) because the expression must be evaluated by the server.

• Modifying STORED column order ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) STORED FIRST, ALGORITHM=COPY; Rebuilds the table in place.

• Dropping a STORED column ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INPLACE, LOCK=NONE; Rebuilds the table in place.

• Adding a VIRTUAL column ALTER TABLE t1 ADD COLUMN (c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL), ALGORITHM=INSTANT;

Adding a virtual column can be performed instantly or in place for non-partitioned tables.

Adding a VIRTUAL is not an in-place operation for partitioned tables.



• Modifying VIRTUAL column order ALTER TABLE t1 MODIFY COLUMN c2 INT GENERATED ALWAYS AS (c1 + 1) VIRTUAL FIRST, ALGORITHM=COPY;

• Dropping a VIRTUAL column ALTER TABLE t1 DROP COLUMN c2, ALGORITHM=INSTANT; Dropping a VIRTUAL column can be performed instantly or in place for non-partitioned tables.

**Foreign** **Key** **Operations**

The following table provides an overview of online DDL support for foreign key operations. An asterisk indicates additional information, an exception, or a dependency. For details, see [Syntax and Usage](#_bookmark63) [Notes](#_bookmark63).

**Table** **15.20** **Online** **DDL** **Support** **for** **Foreign** **Key** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Adding a  foreign key  constraint | No | Yes\* | No | Yes | Yes |
| Dropping a  foreign key  constraint | No | Yes | No | Yes | Yes |

**Syntax** **and** **Usage** **Notes**

• Adding a foreign key constraint

The INPLACE algorithm is supported when foreign\_key\_checks is disabled. Otherwise, only the COPY algorithm is supported.

ALTER TABLE *tbl1* ADD CONSTRAINT *fk\_name* FOREIGN KEY *index* (*col1*)

REFERENCES *tbl2* (*col2*) *referential\_actions*;

• Dropping a foreign key constraint

ALTER TABLE *tbl* DROP FOREIGN KEY *fk\_name*;

Dropping a foreign key can be performed online with the foreign\_key\_checks option enabled or disabled.

If you do not know the names of the foreign key constraints on a particular table, issue the following statement and find the constraint name in the CONSTRAINT clause for each foreign key:

SHOW CREATE TABLE *table*\G

Or, query the Information Schema TABLE\_CONSTRAINTS table and use the CONSTRAINT\_NAME and CONSTRAINT\_TYPE columns to identify the foreign key names.

You can also drop a foreign key and its associated index in a single statement:

ALTER TABLE *table* DROP FOREIGN KEY *constraint*, DROP INDEX *index*;

**Note**

If foreign keys are already present in the table being altered (that is, it is a child table containing a FOREIGN KEY ... REFERENCE clause), additional

|  |  |  |
| --- | --- | --- |
| **Table** **Operations** |  | restrictions apply to online DDL operations, even those not directly involving the foreign key columns:  • An ALTER TABLE on the child table could wait for another transaction to commit, if a change to the parent table causes associated changes in the child table through an ON UPDATE or ON DELETE clause using the CASCADE or SET NULL parameters.  • In the same way, if a table is the parent table in a foreign key relationship, even though it does not contain any FOREIGN KEY clauses, it could wait for the ALTER TABLE to complete if an INSERT, UPDATE, or DELETE statement causes an ON UPDATE or ON DELETE action in the child table. |

The following table provides an overview of online DDL support for table operations. An asterisk indicates additional information, an exception, or a dependency. For details, see [Syntax and Usage](#_bookmark64) [Notes](#_bookmark64).

**Table** **15.21** **Online** **DDL** **Support** **for** **Table** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Changing the  ROW\_FORMAT | No | Yes | Yes | Yes | No |
| Changing the  KEY\_BLOCK\_SI | No  ZE | Yes | Yes | Yes | No |
| Setting persistent table statistics | No | Yes | No | Yes | Yes |
| Specifying a  character set | No | Yes | Yes\* | Yes | No |
| Converting a  character set | No | No | Yes\* | No | No |
| Optimizing a  table | No | Yes\* | Yes | Yes | No |
| Rebuilding with the FORCE option | No | Yes\* | Yes | Yes | No |
| Performing a  null rebuild | No | Yes\* | Yes | Yes | No |
| Renaming a  table | Yes | Yes | No | Yes | Yes |

**Syntax** **and** **Usage** **Notes**

• Changing the ROW\_FORMAT

ALTER TABLE *tbl\_name* ROW\_FORMAT = *row\_format*, ALGORITHM=INPLACE, LOCK=NONE;

Data is reorganized substantially, making it an expensive operation.

For additional information about the ROW\_FORMAT option, see Table Options.

• Changing the KEY\_BLOCK\_SIZE

ALTER TABLE *tbl\_name* KEY\_BLOCK\_SIZE = *value*, ALGORITHM=INPLACE, LOCK=NONE; Data is reorganized substantially, making it an expensive operation.

For additional information about the KEY\_BLOCK\_SIZE option, see Table Options.

• Setting persistent table statistics options ALTER TABLE *tbl\_name* STATS\_PERSISTENT=0, STATS\_SAMPLE\_PAGES=20, STATS\_AUTO\_RECALC=1, ALGORITHM=INPLACE, Only modifies table metadata.

Persistent statistics include STATS\_PERSISTENT, STATS\_AUTO\_RECALC, and STATS\_SAMPLE\_PAGES. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters” .

• Specifying a character set ALTER TABLE *tbl\_name* CHARACTER SET = *charset\_name*, ALGORITHM=INPLACE, LOCK=NONE; Rebuilds the table if the new character encoding is different.

• Converting a character set ALTER TABLE *tbl\_name* CONVERT TO CHARACTER SET *charset\_name*, ALGORITHM=COPY; Rebuilds the table if the new character encoding is different.

• Optimizing a table OPTIMIZE TABLE *tbl\_name*;

In-place operation is not supported for tables with FULLTEXT indexes. The operation uses the INPLACE algorithm, but ALGORITHM and LOCK syntax is not permitted.

• Rebuilding a table with the FORCE option ALTER TABLE *tbl\_name* FORCE, ALGORITHM=INPLACE, LOCK=NONE;

Uses ALGORITHM=INPLACE as of MySQL 5.6.17. ALGORITHM=INPLACE is not supported for tables with FULLTEXT indexes.

• Performing a "null" rebuild ALTER TABLE *tbl\_name* ENGINE=InnoDB, ALGORITHM=INPLACE, LOCK=NONE;

Uses ALGORITHM=INPLACE as of MySQL 5.6.17. ALGORITHM=INPLACE is not supported for tables with FULLTEXT indexes.

• Renaming a table ALTER TABLE *old\_tbl\_name* RENAME TO *new\_tbl\_name*, ALGORITHM=INSTANT;

Renaming a table can be performed instantly or in place. MySQL renames files that correspond to the table *tbl\_name* without making a copy. (You can also use the RENAME TABLE statement to rename tables. See Section 13.1.36, “RENAME TABLE Statement” .) Privileges granted specifically for the renamed table are not migrated to the new name. They must be changed manually.

**Tablespace** **Operations**

The following table provides an overview of online DDL support for tablespace operations. For details, see [Syntax and Usage Notes](#_bookmark65).

**Table** **15.22** **Online** **DDL** **Support** **for** **Tablespace** **Operations**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **Operation** | **Instant** | **In** **Place** | **Rebuilds** **Table** | **Permits**  **Concurrent**  **DML** | **Only** **Modifies** **Metadata** |
| Renaming  a general  tablespace | No | Yes | No | Yes | Yes |
| Enabling or  disabling  general  tablespace  encryption | No | Yes | No | Yes | No |
| Enabling or  disabling file-  per-table  tablespace  encryption | No | No | Yes | No | No |

**Syntax** **and** **Usage** **Notes**

• Renaming a general tablespace ALTER TABLESPACE *tablespace\_name* RENAME TO *new\_tablespace\_name*;

ALTER TABLESPACE ... RENAME TO uses the INPLACE algorithm but does not support the ALGORITHM clause.

• Enabling or disabling general tablespace encryption ALTER TABLESPACE *tablespace\_name* ENCRYPTION='Y';

ALTER TABLESPACE ... ENCRYPTION uses the INPLACE algorithm but does not support the ALGORITHM clause.

For related information, see [Section 15.13, “InnoDB Data-at-Rest Encryption”](#_bookmark66) .

• Enabling or disabling file-per-table tablespace encryption ALTER TABLE *tbl\_name* ENCRYPTION='Y', ALGORITHM=COPY; For related information, see [Section 15.13, “InnoDB Data-at-Rest Encryption”](#_bookmark66) .

**Partitioning** **Operations**

With the exception of some ALTER TABLE partitioning clauses, online DDL operations for partitioned InnoDB tables follow the same rules that apply to regular InnoDB tables.

Some ALTER TABLE partitioning clauses do not go through the same internal online DDL API as regular non-partitioned InnoDB tables. As a result, online support for ALTER TABLE partitioning clauses varies.

The following table shows the online status for each ALTER TABLE partitioning statement. Regardless of the online DDL API that is used, MySQL attempts to minimize data copying and locking where possible.

ALTER TABLE partitioning options that use ALGORITHM=COPY or that only permit “ALGORITHM=DEFAULT, LOCK=DEFAULT”, repartition the table using the COPY algorithm. In other words, a new partitioned table is created with the new partitioning scheme. The newly created table includes any changes applied by the ALTER TABLE statement, and table data is copied into the new table structure.

ALGORITHM=INPLACE,

ALGORITHM=INPLACE,

ALGORITHM=INPLACE,

ALGORITHM=INPLACE

**Table** **15.23** **Online** **DDL** **Support** **for** **Partitioning** **Operations**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Partitioning**  **Clause** | **Instant** | **In** **Place** | **Permits** **DML** | **Notes** |
| PARTITION BY | No | No | No | Permits  ALGORITHM=COPY,  LOCK={DEFAULT |  SHARED |  EXCLUSIVE} |
| ADD PARTITION | No | Yes\* | Yes\* | LOCK={DEFAULT |  NONE |SHARED |  EXCLUSISVE}  is supported  for RANGE and  LIST partitions,  LOCK={DEFAULT |  SHARED |  EXCLUSISVE}  for HASH and KEY partitions, and  ALGORITHM=COPY,  LOCK={SHARED | EXCLUSIVE} for all partition types. Does not copy existing data for tables partitioned by RANGE or LIST. Concurrent queries are permitted with  ALGORITHM=COPY  for tables partitioned by HASH or LIST, as MySQL copies the data while holding a shared lock. |
| DROP PARTITION | No | Yes\* | Yes\* | LOCK={DEFAULT | NONE |SHARED | EXCLUSIVE} is supported. Does not copy data for tables partitioned by RANGE or LIST.  DROP  PARTITION with  deletes data stored in the partition and drops the partition.  However, DROP  PARTITION with  ALGORITHM=COPY |

or

old\_alter\_table=ON

rebuilds the

partitioned table

and attempts to

move data from the

dropped partition

to another partition

with a compatible

PARTITION ...

VALUES definition.

Data that cannot

be moved to

another partition is

deleted.

Only permits

ALGORITHM=DEFAULT, LOCK=DEFAULT

Only permits

ALGORITHM=DEFAULT, LOCK=DEFAULT

ALGORITHM=INPLACE,

LOCK={DEFAULT |

SHARED |

EXCLUSIVE} is

supported.

ALGORITHM=INPLACE,

LOCK={DEFAULT |

SHARED |

EXCLUSIVE} is

supported.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Partitioning**  **Clause** | **Instant** | **In** **Place** | **Permits** **DML** | **Notes** |
|  |  |  |  |  |
| DISCARD  PARTITION | No | No | No |  |
| IMPORT  PARTITION | No | No | No |  |
| TRUNCATE  PARTITION | No | Yes | Yes | Does not copy existing data. It merely deletes rows; it does not alter the definition of the table itself, or of any of its partitions. |
| COALESCE  PARTITION | No | Yes\* | No |  |
| REORGANIZE  PARTITION | No | Yes\* | No |  |
| EXCHANGE  PARTITION | No | Yes | Yes |  |
| ANALYZE  PARTITION | No | Yes | Yes |  |
| CHECK  PARTITION | No | Yes | Yes |  |
| OPTIMIZE  PARTITION | No | No | No | ALGORITHM and LOCK clauses are ignored. Rebuilds the entire table. See  Section 24.3.4, |

ALGORITHM=INPLACE,

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Partitioning**  **Clause** | **Instant** | **In** **Place** | **Permits** **DML** | **Notes** |
|  |  |  |  | “Maintenance of  Partitions” . |
| REBUILD  PARTITION | No | Yes\* | No | LOCK={DEFAULT | SHARED | EXCLUSIVE} is supported. |
| REPAIR  PARTITION | No | Yes | Yes |  |
| REMOVE  PARTITIONING | No | No | No | Permits  ALGORITHM=COPY,  LOCK={DEFAULT |  SHARED |  EXCLUSIVE} |

Non-partitioning online ALTER TABLE operations on partitioned tables follow the same rules that apply to regular tables. However, ALTER TABLE performs online operations on each table partition, which causes increased demand on system resources due to operations being performed on multiple partitions.

For additional information about ALTER TABLE partitioning clauses, see Partitioning Options, and Section 13.1.9.1, “ALTER TABLE Partition Operations” . For information about partitioning in general, see Chapter 24, *Partitioning*.

**15.12.2** **Online** **DDL** **Performance** **and** **Concurrency**

Online DDL improves several aspects of MySQL operation:

• Applications that access the table are more responsive because queries and DML operations on the table can proceed while the DDL operation is in progress. Reduced locking and waiting for MySQL server resources leads to greater scalability, even for operations that are not involved in the DDL operation.

• Instant operations only modify metadata in the data dictionary. An exclusive metadata lock on the table may be taken briefly during the execution phase of the operation. Table data is unaffected, making operations instantaneous. Concurrent DML is permitted.

• Online operations avoid the disk I/O and CPU cycles associated with the table-copy method, which minimizes overall load on the database. Minimizing load helps maintain good performance and high throughput during the DDL operation.

• Online operations read less data into the buffer pool than table-copy operations, which reduces purging of frequently accessed data from memory. Purging of frequently accessed data can cause a temporary performance dip after a DDL operation.

**The** **LOCK** **clause**

By default, MySQL uses as little locking as possible during a DDL operation. The LOCK clause can be specified for in-place operations and some copy operations to enforce more restrictive locking, if required. If the LOCK clause specifies a less restrictive level of locking than is permitted for a particular DDL operation, the statement fails with an error. LOCK clauses are described below, in order of least to most restrictive:

• LOCK=NONE:

Permits concurrent queries and DML.

For example, use this clause for tables involving customer signups or purchases, to avoid making the tables unavailable during lengthy DDL operations.

• LOCK=SHARED:

Permits concurrent queries but blocks DML.

For example, use this clause on data warehouse tables, where you can delay data load operations until the DDL operation is finished, but queries cannot be delayed for long periods.

• LOCK=DEFAULT:

Permits as much concurrency as possible (concurrent queries, DML, or both). Omitting the LOCK clause is the same as specifying LOCK=DEFAULT.

Use this clause when you do not expect the default locking level of the DDL statement to cause any availability problems for the table.

• LOCK=EXCLUSIVE:

Blocks concurrent queries and DML.

Use this clause if the primary concern is finishing the DDL operation in the shortest amount of time possible, and concurrent query and DML access is not necessary. You might also use this clause if the server is supposed to be idle, to avoid unexpected table accesses.

**Online** **DDL** **and** **Metadata** **Locks**

Online DDL operations can be viewed as having three phases:

• *Phase* *1:* *Initialization*

In the initialization phase, the server determines how much concurrency is permitted during the operation, taking into account storage engine capabilities, operations specified in the statement, and user-specified ALGORITHM and LOCK options. During this phase, a shared upgradeable metadata lock is taken to protect the current table definition.

• *Phase* *2:* *Execution*

In this phase, the statement is prepared and executed. Whether the metadata lock is upgraded to exclusive depends on the factors assessed in the initialization phase. If an exclusive metadata lock is required, it is only taken briefly during statement preparation.

• *Phase* *3:* *Commit* *Table* *Definition*

In the commit table definition phase, the metadata lock is upgraded to exclusive to evict the old table definition and commit the new one. Once granted, the duration of the exclusive metadata lock is brief.

Due to the exclusive metadata lock requirements outlined above, an online DDL operation may have to wait for concurrent transactions that hold metadata locks on the table to commit or rollback. Transactions started before or during the DDL operation can hold metadata locks on the table being altered. In the case of a long running or inactive transaction, an online DDL operation can time out waiting for an exclusive metadata lock. Additionally, a pending exclusive metadata lock requested by an online DDL operation blocks subsequent transactions on the table.

The following example demonstrates an online DDL operation waiting for an exclusive metadata lock, and how a pending metadata lock blocks subsequent transactions on the table.

Session 1:

mysql> CREATE TABLE t1 (c1 INT) ENGINE=InnoDB;

mysql> START TRANSACTION;

mysql> SELECT \* FROM t1;

The session 1 SELECT statement takes a shared metadata lock on table t1.

Session 2:

mysql> ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE;

The online DDL operation in session 2, which requires an exclusive metadata lock on table t1 to commit table definition changes, must wait for the session 1 transaction to commit or roll back.

Session 3:

mysql> SELECT \* FROM t1;

The SELECT statement issued in session 3 is blocked waiting for the exclusive metadata lock requested by the ALTER TABLE operation in session 2 to be granted.

You can use SHOW FULL PROCESSLIST to determine if transactions are waiting for a metadata lock.

mysql> **SHOW** **FULL** **PROCESSLIST\G**

...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 5

User: root

Host: localhost

db: test

Command: Query

Time: 44

State: Waiting for table metadata lock

Info: ALTER TABLE t1 ADD COLUMN x INT, ALGORITHM=INPLACE, LOCK=NONE

...

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 4. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Id: 7

User: root

Host: localhost

db: test

Command: Query

Time: 5

State: Waiting for table metadata lock

Info: SELECT \* FROM t1

4 rows in set (0.00 sec)

Metadata lock information is also exposed through the Performance Schema metadata\_locks table, which provides information about metadata lock dependencies between sessions, the metadata lock a session is waiting for, and the session that currently holds the metadata lock. For more information, see Section 27.12.13.3, “The metadata\_locks Table” .

**Online** **DDL** **Performance**

The performance of a DDL operation is largely determined by whether the operation is performed instantly, in place, and whether it rebuilds the table.

To assess the relative performance of a DDL operation, you can compare results using ALGORITHM=INSTANT, ALGORITHM=INPLACE, and ALGORITHM=COPY. A statement can also be run with old\_alter\_table enabled to force the use of ALGORITHM=COPY.

For DDL operations that modify table data, you can determine whether a DDL operation performs changes in place or performs a table copy by looking at the “rows affected” value displayed after the command finishes. For example:

• Changing the default value of a column (fast, does not affect the table data): Query OK, 0 rows affected (0.07 sec)

• Adding an index (takes time, but 0 rows affected shows that the table is not copied): Query OK, 0 rows affected (21.42 sec)



• Changing the data type of a column (takes substantial time and requires rebuilding all the rows of the table):

Query OK, 1671168 rows affected (1 min 35.54 sec)

Before running a DDL operation on a large table, check whether the operation is fast or slow as follows:

1. Clone the table structure.

2. Populate the cloned table with a small amount of data.

3. Run the DDL operation on the cloned table.

4. Check whether the “rows affected” value is zero or not. A nonzero value means the operation copies table data, which might require special planning. For example, you might do the DDL operation during a period of scheduled downtime, or on each replica server one at a time.

**Note**

For a greater understanding of the MySQL processing associated with a DDL operation, examine Performance Schema and INFORMATION\_SCHEMA tables related to InnoDB before and after DDL operations to see the number of physical reads, writes, memory allocations, and so on.

Performance Schema stage events can be used to monitor ALTER TABLE progress. See [Section 15.16.1, “Monitoring ALTER TABLE Progress for InnoDB](#_bookmark68) [Tables Using Performance Schema”](#_bookmark68) .

Because there is some processing work involved with recording the changes made by concurrent DML operations, then applying those changes at the end, an online DDL operation could take longer overall than the table-copy mechanism that blocks table access from other sessions. The reduction in raw performance is balanced against better responsiveness for applications that use the table. When evaluating the techniques for changing table structure, consider end-user perception of performance, based on factors such as load times for web pages.

**15.12.3** **Online** **DDL** **Space** **Requirements**

Disk space requirements for online DDL operations are outlined below. The requirements do not apply to operations that are performed instantly.

• Temporary log files:

A temporary log file records concurrent DML when an online DDL operation creates an index or alters a table. The temporary log file is extended as required

by the value of [innodb\_sort\_buffer\_size](#_bookmark70) up to a maximum specified by [innodb\_online\_alter\_log\_max\_size](#_bookmark71). If the operation takes a long time and concurrent DML modifies the table so much that the size of the temporary log file exceeds the value of [innodb\_online\_alter\_log\_max\_size](#_bookmark71), the online DDL operation fails with a DB\_ONLINE\_LOG\_TOO\_BIG error, and uncommitted concurrent DML operations are rolled back. A large [innodb\_online\_alter\_log\_max\_size](#_bookmark71) setting permits more DML during an online DDL operation, but it also extends the period of time at the end of the DDL operation when the table is locked to apply logged DML.

The [innodb\_sort\_buffer\_size](#_bookmark70) variable also defines the size of the temporary log file read buffer and write buffer.

• Temporary sort files:

Online DDL operations that rebuild the table write temporary sort files to the MySQL temporary directory ($TMPDIR on Unix, %TEMP% on Windows, or the directory specified by --tmpdir) during index creation. Temporary sort files are not created in the directory that contains the original table. Each temporary sort file is large enough to hold one column of data, and each sort file is removed

when its data is merged into the final table or index. Operations involving temporary sort files may require temporary space equal to the amount of data in the table plus indexes. An error is reported if online DDL operation uses all of the available disk space on the file system where the data directory resides.

If the MySQL temporary directory is not large enough to hold the sort files, set tmpdir to a different directory. Alternatively, define a separate temporary directory for online DDL operations using [innodb\_tmpdir](#_bookmark72). This option was introduced to help avoid temporary directory overflows that could occur as a result of large temporary sort files.

• Intermediate table files:

Some online DDL operations that rebuild the table create a temporary intermediate table file in the same directory as the original table. An intermediate table file may require space equal to the size of the original table. Intermediate table file names begin with #sql-ib prefix and only appear briefly during the online DDL operation.

The [innodb\_tmpdir](#_bookmark72) option is not applicable to intermediate table files.

**15.12.4** **Online** **DDL** **Memory** **Management**

Online DDL operations that create or rebuild secondary indexes allocate temporary buffers during different phases of index creation. The [innodb\_ddl\_buffer\_size](#_bookmark74) variable, introduced in MySQL 8.0.27, defines the maximum buffer size for online DDL operations. The default setting is 1048576 bytes (1 MB). The setting applies to buffers created by threads executing online DDL operations. Defining an appropriate buffer size limit avoids potential out of memory errors for online DDL operations that create or rebuild secondary indexes. The maximum buffer size per DDL thread is the maximum buffer size divided by the number of DDL threads ([innodb\_ddl\_buffer\_size](#_bookmark74)/[innodb\_ddl\_threads](#_bookmark75)).

Prior to MySQL 8.0.27, [innodb\_sort\_buffer\_size](#_bookmark70) variable defines the buffer size for online DDL operations that create or rebuild secondary indexes.

**15.12.5** **Configuring** **Parallel** **Threads** **for** **Online** **DDL** **Operations**

The workflow of an online DDL operation that creates or rebuilds a secondary index involves:

• Scanning the clustered index and writing data to temporary sort files

• Sorting the data

• Loading sorted data from the temporary sort files into the secondary index

The number of parallel threads that can be used to scan clustered index is defined by the [innodb\_parallel\_read\_threads](#_bookmark77) variable. The default setting is 4. The maximum setting is 256, which is the maximum number for all sessions. The actual number of threads that scan the clustered index is the number defined by the [innodb\_parallel\_read\_threads](#_bookmark77) setting or the number of index subtrees to scan, whichever is smaller. If the thread limit is reached, sessions fall back to using a single thread.

The number of parallel threads that sort and load data is controlled by the [innodb\_ddl\_threads](#_bookmark75) variable, introduced in MySQL 8.0.27. The default setting is 4. Prior to MySQL 8.0.27, sort and load operations are single-threaded.

The following limitations apply:

• Parallel threads are not supported for building indexes that include virtual columns.

• Parallel threads are not supported for full-text index creation.

• Parallel threads are not supported for spatial index creation.

• Parallel scan is not supported on tables defined with virtual columns.

• Parallel scan is not supported on tables defined with a full-text index.

• Parallel scan is not supported on tables defined with a spatial index.

**15.12.6** **Simplifying** **DDL** **Statements** **with** **Online** **DDL**

Before the introduction of online DDL, it was common practice to combine many DDL operations into a single ALTER TABLE statement. Because each ALTER TABLE statement involved copying and rebuilding the table, it was more efficient to make several changes to the same table at once, since those changes could all be done with a single rebuild operation for the table. The downside was that SQL code involving DDL operations was harder to maintain and to reuse in different scripts. If the specific changes were different each time, you might have to construct a new complex ALTER TABLE for each slightly different scenario.

For DDL operations that can be done online, you can separate them into individual ALTER TABLE statements for easier scripting and maintenance, without sacrificing efficiency. For example, you might take a complicated statement such as:

ALTER TABLE t1 ADD INDEX i1(c1), ADD UNIQUE INDEX i2(c2),

CHANGE c4\_old\_name c4\_new\_name INTEGER UNSIGNED;

and break it down into simpler parts that can be tested and performed independently, such as:

ALTER TABLE t1 ADD INDEX i1(c1);

ALTER TABLE t1 ADD UNIQUE INDEX i2(c2);

ALTER TABLE t1 CHANGE c4\_old\_name c4\_new\_name INTEGER UNSIGNED NOT NULL;

You might still use multi-part ALTER TABLE statements for:

• Operations that must be performed in a specific sequence, such as creating an index followed by a foreign key constraint that uses that index.

• Operations all using the same specific LOCK clause, that you want to either succeed or fail as a group.

• Operations that cannot be performed online, that is, that still use the table-copy method.

• Operations for which you specify ALGORITHM=COPY or old\_alter\_table=1, to force the table- copying behavior if needed for precise backward-compatibility in specialized scenarios.

**15.12.7** **Online** **DDL** **Failure** **Conditions**

The failure of an online DDL operation is typically due to one of the following conditions:

• An ALGORITHM clause specifies an algorithm that is not compatible with the particular type of DDL operation or storage engine.

• A LOCK clause specifies a low degree of locking (SHARED or NONE) that is not compatible with the particular type of DDL operation.

• A timeout occurs while waiting for an exclusive lock on the table, which may be needed briefly during the initial and final phases of the DDL operation.

• The tmpdir or [innodb\_tmpdir](#_bookmark72) file system runs out of disk space, while MySQL writes temporary sort files on disk during index creation. For more information, see [Section 15.12.3, “Online DDL](#_bookmark69)

[Space Requirements”](#_bookmark69) .

• The operation takes a long time and concurrent DML modifies the table so much that the size of the temporary online log exceeds the value of the [innodb\_online\_alter\_log\_max\_size](#_bookmark71) configuration option. This condition causes a DB\_ONLINE\_LOG\_TOO\_BIG error.

• Concurrent DML makes changes to the table that are allowed with the original table definition, but not with the new one. The operation only fails at the very end, when MySQL tries to apply all the changes from concurrent DML statements. For example, you might insert duplicate values into

a column while a unique index is being created, or you might insert NULL values into a column while creating a primary key index on that column. The changes made by the concurrent DML take precedence, and the ALTER TABLE operation is effectively rolled back.

**15.12.8** **Online** **DDL** **Limitations**

The following limitations apply to online DDL operations:

• The table is copied when creating an index on a TEMPORARY TABLE.

• The ALTER TABLE clause LOCK=NONE is not permitted if there are ON...CASCADE or ON...SET NULL constraints on the table.

• Before an in-place online DDL operation can finish, it must wait for transactions that hold metadata locks on the table to commit or roll back. An online DDL operation may briefly require an exclusive metadata lock on the table during its execution phase, and always requires one in the final phase of the operation when updating the table definition. Consequently, transactions holding metadata locks on the table can cause an online DDL operation to block. The transactions that hold metadata locks on the table may have been started before or during the online DDL operation. A long running or inactive transaction that holds a metadata lock on the table can cause an online DDL operation to timeout.

• When running an in-place online DDL operation, the thread that runs the ALTER TABLE statement applies an online log of DML operations that were run concurrently on the same table from other connection threads. When the DML operations are applied, it is possible to encounter a duplicate key entry error (ERROR 1062 (23000): Duplicate entry), even if the duplicate entry is only temporary and would be reverted by a later entry in the online log. This is similar to the idea of a

foreign key constraint check in InnoDB in which constraints must hold during a transaction.

• OPTIMIZE TABLE for an InnoDB table is mapped to an ALTER TABLE operation to rebuild the table and update index statistics and free unused space in the clustered index. Secondary indexes are not created as efficiently because keys are inserted in the order they appeared in the primary key. OPTIMIZE TABLE is supported with the addition of online DDL support for rebuilding regular and partitioned InnoDB tables.

• Tables created before MySQL 5.6 that include temporal columns (DATE, DATETIME or TIMESTAMP) and have not been rebuilt using ALGORITHM=COPY do not support ALGORITHM=INPLACE. In this case, an ALTER TABLE ... ALGORITHM=INPLACE operation returns the following error:

ERROR 1846 (0A000): ALGORITHM=INPLACE is not supported.

Reason: Cannot change column type INPLACE. Try ALGORITHM=COPY.

• The following limitations are generally applicable to online DDL operations on large tables that involve rebuilding the table:

• There is no mechanism to pause an online DDL operation or to throttle I/O or CPU usage for an online DDL operation.

• Rollback of an online DDL operation can be expensive should the operation fail.

• Long running online DDL operations can cause replication lag. An online DDL operation must finish running on the source before it is run on the replica. Also, DML that was processed concurrently on the source is only processed on the replica after the DDL operation on the replica is completed.

For additional information related to running online DDL operations on large tables, see [Section 15.12.2, “Online DDL Performance and Concurrency”](#_bookmark67) .

**15.13** **InnoDB** **Data-at-Rest** **Encryption**

InnoDB supports data-at-rest encryption for file-per-table tablespaces, general tablespaces, the mysql system tablespace, redo logs, and undo logs.

As of MySQL 8.0.16, setting an encryption default for schemas and general tablespaces is also supported, which permits DBAs to control whether tables created in those schemas and tablespaces are encrypted.

InnoDB data-at-rest encryption features and capabilities are described under the following topics in this section.

• [About Data-at-Rest Encryption](#_bookmark78)

• [Encryption Prerequisites](#_bookmark79)

• [Defining an Encryption Default for Schemas and General Tablespaces](#_bookmark80)

• [File-Per-Table Tablespace Encryption](#_bookmark81)

• [General Tablespace Encryption](#_bookmark82)

• [Doublewrite File Encryption](#_bookmark83)

• [mysql System Tablespace Encryption](#_bookmark84)

• [Redo Log Encryption](#_bookmark85)

• [Undo Log Encryption](#_bookmark86)

• [Master Key Rotation](#_bookmark87)

• [Encryption and Recovery](#_bookmark88)

• [Exporting Encrypted Tablespaces](#_bookmark89)

• [Encryption and Replication](#_bookmark90)

• [Identifying Encrypted Tablespaces and Schemas](#_bookmark91)

• [Monitoring Encryption Progress](#_bookmark92)

• [Encryption Usage Notes](#_bookmark93)

• [Encryption Limitations](#_bookmark94)

**About** **Data-at-Rest** **Encryption**

InnoDB uses a two tier encryption key architecture, consisting of a master encryption key and tablespace keys. When a tablespace is encrypted, a tablespace key is encrypted and stored in the tablespace header. When an application or authenticated user wants to access encrypted tablespace data, InnoDB uses a master encryption key to decrypt the tablespace key. The decrypted version of a tablespace key never changes, but the master encryption key can be changed as required. This action is referred to as *master* *key* *rotation*.

The data-at-rest encryption feature relies on a keyring component or plugin for master encryption key management.

All MySQL editions provide a component\_keyring\_file component and keyring\_file plugin, each of which stores keyring data in a file local to the server host.

MySQL Enterprise Edition offers additional keyring components and plugins:

• component\_keyring\_encrypted\_file: Stores keyring data in an encrypted, password- protected file local to the server host.

• keyring\_encrypted\_file: Stores keyring data in an encrypted, password-protected file local to the server host.

• keyring\_okv: A KMIP 1.1 plugin for use with KMIP-compatible back end keyring storage products. Supported KMIP-compatible products include centralized key management solutions such as



Oracle Key Vault, Gemalto KeySecure, Thales Vormetric key management server, and Fornetix Key

Orchestration.

• keyring\_aws: Communicates with the Amazon Web Services Key Management Service (AWS KMS) as a back end for key generation and uses a local file for key storage.

• keyring\_hashicorp: Communicates with HashiCorp Vault for back end storage.

**Warning**

For encryption key management, the component\_keyring\_file and component\_keyring\_encrypted\_file components, and the keyring\_file and keyring\_encrypted\_file plugins are not intended as a regulatory compliance solution. Security standards such as PCI, FIPS, and others require use of key management systems to secure, manage, and protect encryption keys in key vaults or hardware security modules (HSMs).

A secure and robust encryption key management solution is critical for security and for compliance with various security standards. When the data-at-rest encryption feature uses a centralized key management solution, the feature is referred to as “MySQL Enterprise Transparent Data Encryption (TDE)” .

The data-at-rest encryption feature supports the Advanced Encryption Standard (AES) block-based encryption algorithm. It uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption.

For frequently asked questions about the data-at-rest encryption feature, see Section A.17, “MySQL 8.0 FAQ: InnoDB Data-at-Rest Encryption” .

**Encryption** **Prerequisites**

• A keyring component or plugin must be installed and configured at startup. Early loading ensures that the component or plugin is available prior to initialization of the InnoDB storage engine. For keyring installation and configuration instructions, see Section 6.4.4, “The MySQL Keyring” . The instructions show how to ensure that the chosen component or plugin is active.

Only one keyring component or plugin should be enabled at a time. Enabling multiple keyring components or plugins is unsupported and results may not be as anticipated.

**Important**

Once encrypted tablespaces are created in a MySQL instance, the keyring component or plugin that was loaded when creating the encrypted tablespace must continue to be loaded at startup. Failing to do so results in errors when starting the server and during InnoDB recovery.

• When encrypting production data, ensure that you take steps to prevent loss of the master encryption key. *If* *the* *master* *encryption* *key* *is* *lost,* *data* *stored* *in* *encrypted* *tablespace* *files* *is* *unrecoverable.* If you use the component\_keyring\_file or

component\_keyring\_encrypted\_file component, or the keyring\_file or keyring\_encrypted\_file plugin, create a backup of the keyring data file immediately after creating the first encrypted tablespace, before master key rotation, and after master key rotation. For each component, its configuration file indicates the data file location. The keyring\_file\_data configuration option defines the keyring data file location for the keyring\_file plugin. The keyring\_encrypted\_file\_data configuration option defines the keyring data file location for the keyring\_encrypted\_file plugin. If you use the keyring\_okv or keyring\_aws plugin, ensure that you have performed the necessary configuration. For instructions, see Section 6.4.4, “The MySQL Keyring” .

**Defining** **an** **Encryption** **Default** **for** **Schemas** **and** **General** **Tablespaces**

As of MySQL 8.0.16, the default\_table\_encryption system variable defines the default encryption setting for schemas and general tablespaces. CREATE TABLESPACE and CREATE SCHEMA operations apply the default\_table\_encryption setting when an ENCRYPTION clause is not specified explicitly.

ALTER SCHEMA and ALTER TABLESPACE operations do not apply the default\_table\_encryption setting. An ENCRYPTION clause must be specified explicitly to alter the encryption of an existing schema or general tablespace.

The default\_table\_encryption variable can be set for an individual client connection or globally using SET syntax. For example, the following statement enables default schema and tablespace encryption globally:

mysql> SET GLOBAL default\_table\_encryption=ON;

The default encryption setting for a schema can also be defined using the DEFAULT ENCRYPTION clause when creating or altering a schema, as in this example:

mysql> CREATE SCHEMA test DEFAULT ENCRYPTION = 'Y';

If the DEFAULT ENCRYPTION clause is not specified when creating a schema, the default\_table\_encryption setting is applied. The DEFAULT ENCRYPTION clause must be specified to alter the default encryption of an existing schema. Otherwise, the schema retains its current encryption setting.

By default, a table inherits the encryption setting of the schema or general tablespace it is created in. For example, a table created in an encryption-enabled schema is encrypted by default. This behavior enables a DBA to control table encryption usage by defining and enforcing schema and general tablespace encryption defaults.

Encryption defaults are enforced by enabling the table\_encryption\_privilege\_check system variable. When table\_encryption\_privilege\_check is enabled, a privilege check occurs when creating or altering a schema or general tablespace with an encryption setting that differs from the default\_table\_encryption setting, or when creating or altering a table with an encryption setting that differs from the default schema encryption. When table\_encryption\_privilege\_check is disabled (the default), the privilege check does not occur and the previously mentioned operations are permitted to proceed with a warning.

The TABLE\_ENCRYPTION\_ADMIN privilege is required to override default encryption settings when table\_encryption\_privilege\_check is enabled. A DBA can grant this privilege to enable a user to deviate from the default\_table\_encryption setting when creating or altering a schema or general tablespace, or to deviate from the default schema encryption when creating or altering a table. This privilege does not permit deviating from the encryption of a general tablespace when creating or altering a table. A table must have the same encryption setting as the general tablespace it resides in.

**File-Per-Table** **Tablespace** **Encryption**

As of MySQL 8.0.16, a file-per-table tablespace inherits the default encryption of the schema in which the table is created unless an ENCRYPTION clause is specified explicitly in the CREATE TABLE statement. Prior to MySQL 8.0.16, the ENCRYPTION clause must be specified to enable encryption.

mysql> **CREATE** **TABLE** **t1** **(c1** **INT)** **ENCRYPTION** **=** **'Y';**

To alter the encryption of an existing file-per-table tablespace, an ENCRYPTION clause must be specified.

mysql> **ALTER** **TABLE** **t1** **ENCRYPTION** **=** **'Y';**

As of MySQL 8.0.16, if the table\_encryption\_privilege\_check variable is enabled, specifying an ENCRYPTION clause with a setting that differs from the default schema encryption requires the TABLE\_ENCRYPTION\_ADMIN privilege. See [Defining an Encryption Default for Schemas and General](#_bookmark80) [Tablespaces](#_bookmark80).



**General** **Tablespace** **Encryption**

As of MySQL 8.0.16, the default\_table\_encryption variable determines the encryption of a newly created general tablespace unless an ENCRYPTION clause is specified explicitly in the CREATE TABLESPACE statement. Prior to MySQL 8.0.16, an ENCRYPTION clause must be specified to enable encryption.

mysql> **CREATE** **TABLESPACE** **`ts1`** **ADD** **DATAFILE** **'ts1.ibd'** **ENCRYPTION** **=** **'Y'** **Engine=InnoDB;** To alter the encryption of an existing general tablespace, an ENCRYPTION clause must be specified. mysql> **ALTER** **TABLESPACE** **ts1** **ENCRYPTION** **=** **'Y';**

As of MySQL 8.0.16, if the table\_encryption\_privilege\_check variable is enabled, specifying an ENCRYPTION clause with a setting that differs from the default\_table\_encryption setting requires the TABLE\_ENCRYPTION\_ADMIN privilege. See [Defining an Encryption Default for Schemas](#_bookmark80) [and General Tablespaces](#_bookmark80).

**Doublewrite** **File** **Encryption**

Encryption support for doublewrite files is available as of MySQL 8.0.23. InnoDB automatically encrypts doublewrite file pages that belong to encrypted tablespaces. No action is required. Doublewrite file pages are encrypted using the encryption key of the associated tablespace. The same encrypted page written to a tablespace data file is also written to a doublewrite file. Doublewrite file pages that belong to an unencrypted tablespace remain unencrypted.

During recovery, encrypted doublewrite file pages are unencrypted and checked for corruption.

**mysql** **System** **Tablespace** **Encryption**

Encryption support for the mysql system tablespace is available as of MySQL 8.0.16.

The mysql system tablespace contains the mysql system database and MySQL data dictionary tables. It is unencrypted by default. To enable encryption for the mysql system tablespace, specify the tablespace name and the ENCRYPTION option in an ALTER TABLESPACE statement.

mysql> ALTER TABLESPACE mysql ENCRYPTION = 'Y';

To disable encryption for the mysql system tablespace, set ENCRYPTION = 'N' using an ALTER

TABLESPACE statement. mysql> ALTER TABLESPACE mysql ENCRYPTION = 'N';

Enabling or disabling encryption for the mysql system tablespace requires the CREATE TABLESPACE privilege on all tables in the instance (CREATE TABLESPACE on \*.\*).

**Redo** **Log** **Encryption**

Redo log data encryption is enabled using the [innodb\_redo\_log\_encrypt](#_bookmark95) configuration option. Redo log encryption is disabled by default.

As with tablespace data, redo log data encryption occurs when redo log data is written to disk, and decryption occurs when redo log data is read from disk. Once redo log data is read into memory, it is in unencrypted form. Redo log data is encrypted and decrypted using the tablespace encryption key.

When [innodb\_redo\_log\_encrypt](#_bookmark95) is enabled, unencrypted redo log pages that are present on disk remain unencrypted, and new redo log pages are written to disk in encrypted form. Likewise, when [innodb\_redo\_log\_encrypt](#_bookmark95) is disabled, encrypted redo log pages that are present on disk remain encrypted, and new redo log pages are written to disk in unencrypted form.

**Warning**

A regression introduced in MySQL 8.0.30 prevents disabling redo log encryption once it is enabled. (Bug #108052, Bug #34456802).



From MySQL 8.0.30, redo log encryption metadata, including the tablespace encryption key, is stored in the header of the redo log file with the most recent checkpoint LSN. Before MySQL 8.0.30, redo log encryption metadata, including the tablespace encryption key, is stored in the header of the first redo log file (ib\_logfile0). If the redo log file with the encryption metadata is removed, redo log encryption is disabled.

Once redo log encryption is enabled, a normal restart without the keyring component or plugin or without the encryption key is not possible, as InnoDB must be able to scan redo pages during startup, which is not possible if redo log pages are encrypted. Without the keyring component or plugin or the encryption key, only a forced startup without the redo logs (SRV\_FORCE\_NO\_LOG\_REDO) is possible. See Section 15.21.3, “Forcing InnoDB Recovery” .

**Undo** **Log** **Encryption**

Undo log data encryption is enabled using the [innodb\_undo\_log\_encrypt](#_bookmark96) configuration option. Undo log encryption applies to undo logs that reside in undo tablespaces. See Section 15.6.3.4, “Undo Tablespaces” . Undo log data encryption is disabled by default.

As with tablespace data, undo log data encryption occurs when undo log data is written to disk, and decryption occurs when undo log data is read from disk. Once undo log data is read into memory, it is in unencrypted form. Undo log data is encrypted and decrypted using the tablespace encryption key.

When [innodb\_undo\_log\_encrypt](#_bookmark96) is enabled, unencrypted undo log pages that are present on disk remain unencrypted, and new undo log pages are written to disk in encrypted form. Likewise, when [innodb\_undo\_log\_encrypt](#_bookmark96) is disabled, encrypted undo log pages that are present on disk remain encrypted, and new undo log pages are written to disk in unencrypted form.

Undo log encryption metadata, including the tablespace encryption key, is stored in the header of the undo log file.

**Note**

When undo log encryption is disabled, the server continues to require the keyring component or plugin that was used to encrypt undo log data until the undo tablespaces that contained the encrypted undo log data are truncated. (An encryption header is only removed from an undo tablespace when the undo tablespace is truncated.) For information about truncating undo tablespaces, see Truncating Undo Tablespaces.

**Master** **Key** **Rotation**

The master encryption key should be rotated periodically and whenever you suspect that the key has been compromised.

Master key rotation is an atomic, instance-level operation. Each time the master encryption key is rotated, all tablespace keys in the MySQL instance are re-encrypted and saved back to their respective tablespace headers. As an atomic operation, re-encryption must succeed for all tablespace keys once a rotation operation is initiated. If master key rotation is interrupted by a server failure, InnoDB rolls the operation forward on server restart. For more information, see [Encryption and Recovery](#_bookmark88).

Rotating the master encryption key only changes the master encryption key and re-encrypts tablespace keys. It does not decrypt or re-encrypt associated tablespace data.

Rotating the master encryption key requires the ENCRYPTION\_KEY\_ADMIN privilege (or the deprecated SUPER privilege).

To rotate the master encryption key, run:

mysql> **ALTER** **INSTANCE** **ROTATE** **INNODB** **MASTER** **KEY;**

ALTER INSTANCE ROTATE INNODB MASTER KEY supports concurrent DML. However, it cannot be run concurrently with tablespace encryption operations, and locks are taken to prevent conflicts

that could arise from concurrent execution. If an ALTER INSTANCE ROTATE INNODB MASTER KEY operation is running, it must finish before a tablespace encryption operation can proceed, and vice versa.

**Encryption** **and** **Recovery**

If a server failure occurs during an encryption operation, the operation is rolled forward when the server is restarted. For general tablespaces, the encryption operation is resumed in a background thread from the last processed page.

If a server failure occurs during master key rotation, InnoDB continues the operation on server restart.

The keyring component or plugin must be loaded prior to storage engine initialization so that the information necessary to decrypt tablespace data pages can be retrieved from tablespace headers before InnoDB initialization and recovery activities access tablespace data. (See [Encryption](#_bookmark79) [Prerequisites](#_bookmark79).)

When InnoDB initialization and recovery begin, the master key rotation operation resumes. Due to the server failure, some tablespace keys may already be encrypted using the new master encryption key. InnoDB reads the encryption data from each tablespace header, and if the data indicates that the tablespace key is encrypted using the old master encryption key, InnoDB retrieves the old key from the keyring and uses it to decrypt the tablespace key. InnoDB then re-encrypts the tablespace key using the new master encryption key and saves the re-encrypted tablespace key back to the tablespace header.

**Exporting** **Encrypted** **Tablespaces**

Tablespace export is only supported for file-per-table tablespaces.

When an encrypted tablespace is exported, InnoDB generates a *transfer* *key* that is used to encrypt the tablespace key. The encrypted tablespace key and transfer key are stored in a *tablespace\_name*.cfp file. This file together with the encrypted tablespace file is required to perform an import operation. On import, InnoDB uses the transfer key to decrypt the tablespace key in the *tablespace\_name*.cfp file. For related information, see Section 15.6.1.3, “Importing InnoDB

Tables” .

**Encryption** **and** **Replication**

• The ALTER INSTANCE ROTATE INNODB MASTER KEY statement is only supported in replication environments where the source and replica run a version of MySQL that supports tablespace encryption.

• Successful ALTER INSTANCE ROTATE INNODB MASTER KEY statements are written to the binary log for replication on replicas.

• If an ALTER INSTANCE ROTATE INNODB MASTER KEY statement fails, it is not logged to the binary log and is not replicated on replicas.

• Replication of an ALTER INSTANCE ROTATE INNODB MASTER KEY operation fails if the keyring component or plugin is installed on the source but not on the replica.

• If the keyring\_file or keyring\_encrypted\_file plugin is installed on both the source and a replica but the replica does not have a keyring data file, the replicated ALTER INSTANCE ROTATE INNODB MASTER KEY statement creates the keyring data file on the replica, assuming the keyring file data is not cached in memory. ALTER INSTANCE ROTATE INNODB MASTER KEY uses keyring file data that is cached in memory, if available.

**Identifying** **Encrypted** **Tablespaces** **and** **Schemas**

The Information Schema INNODB\_TABLESPACES table, introduced in MySQL 8.0.13, includes an ENCRYPTION column that can be used to identify encrypted tablespaces.

**WHERE** **CREATE\_OPTIONS** **LIKE** **'%ENCRYPTION%';**

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

| TABLE\_SCHEMA | TABLE\_NAME | CREATE\_OPTIONS |

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

| test | t1 | ENCRYPTION="Y" |

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

**WHERE** **DEFAULT\_ENCRYPTION='YES';**

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

| SCHEMA\_NAME | DEFAULT\_ENCRYPTION |

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

| test | YES |

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

mysql> **SELECT** **SPACE,** **NAME,** **SPACE\_TYPE,** **ENCRYPTION** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLESPACES**

**WHERE** **ENCRYPTION='Y'\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: 4294967294

NAME: mysql

SPACE\_TYPE: General

ENCRYPTION: Y

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: 2

NAME: test/t1

SPACE\_TYPE: Single

ENCRYPTION: Y

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: 3

NAME: ts1

SPACE\_TYPE: General

ENCRYPTION: Y

When the ENCRYPTION option is specified in a CREATE TABLE or ALTER TABLE statement, it is recorded in the CREATE\_OPTIONS column of INFORMATION\_SCHEMA.TABLES. This column can be queried to identify tables that reside in encrypted file-per-table tablespaces.

mysql> **SELECT** **TABLE\_SCHEMA,** **TABLE\_NAME,** **CREATE\_OPTIONS** **FROM** **INFORMATION\_SCHEMA.TABLES**

Query the Information Schema INNODB\_TABLESPACES table to retrieve information about the tablespace associated with a particular schema and table.

mysql> **SELECT** **SPACE,** **NAME,** **SPACE\_TYPE** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLESPACES** **WHERE** **NAME='test/t1';**

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

| SPACE | NAME | SPACE\_TYPE |

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

| 3 | test/t1 | Single |

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

You can identify encryption-enabled schemas by querying the Information Schema SCHEMATA table.

mysql> **SELECT** **SCHEMA\_NAME,** **DEFAULT\_ENCRYPTION** **FROM** **INFORMATION\_SCHEMA.SCHEMATA**

SHOW CREATE SCHEMA also shows the DEFAULT ENCRYPTION clause.

**Monitoring** **Encryption** **Progress**

You can monitor general tablespace and mysql system tablespace encryption progress using

Performance Schema.

The stage/innodb/alter tablespace (encryption) stage event instrument reports WORK\_ESTIMATED and WORK\_COMPLETED information for general tablespace encryption operations.

The following example demonstrates how to enable the stage/innodb/alter tablespace (encryption) stage event instrument and related consumer tables to monitor general tablespace or mysql system tablespace encryption progress. For information about Performance Schema stage event instruments and related consumers, see Section 27.12.5, “Performance Schema Stage Event

Tables” .

1. Enable the stage/innodb/alter tablespace (encryption) instrument:

**WHERE** **NAME** **LIKE** **'stage/innodb/alter** **tablespace** **(encryption)';**

mysql> **USE** **performance\_schema;**

mysql> **UPDATE** **setup\_instruments** **SET** **ENABLED** **=** **'YES'**

2. Enable the stage event consumer tables, which include events\_stages\_current, events\_stages\_history, and events\_stages\_history\_long.

mysql> **UPDATE** **setup\_consumers** **SET** **ENABLED** **=** **'YES'** **WHERE** **NAME** **LIKE** **'%stages%';**

3. Run a tablespace encryption operation. In this example, a general tablespace named ts1 is

encrypted. mysql> **ALTER** **TABLESPACE** **ts1** **ENCRYPTION** **=** **'Y';**

4. Check the progress of the encryption operation by querying the Performance Schema events\_stages\_current table. WORK\_ESTIMATED reports the total number of pages in the tablespace. WORK\_COMPLETED reports the number of pages processed.

mysql> **SELECT** **EVENT\_NAME,** **WORK\_ESTIMATED,** **WORK\_COMPLETED** **FROM** **events\_stages\_current;**

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

| EVENT\_NAME | WORK\_COMPLETED | WORK\_ESTIMATED |

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

| stage/innodb/alter tablespace (encryption) | 1056 | 1407 |

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

The events\_stages\_current table returns an empty set if the encryption operation has completed. In this case, you can check the events\_stages\_history table to view event data for the completed operation. For example:

mysql> **SELECT** **EVENT\_NAME,** **WORK\_COMPLETED,** **WORK\_ESTIMATED** **FROM** **events\_stages\_history;**

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

| EVENT\_NAME | WORK\_COMPLETED | WORK\_ESTIMATED |

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

| stage/innodb/alter tablespace (encryption) | 1407 | 1407 |

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

**Encryption** **Usage** **Notes**

• Plan appropriately when altering an existing file-per-table tablespace with the ENCRYPTION option. Tables residing in file-per-table tablespaces are rebuilt using the COPY algorithm. The INPLACE algorithm is used when altering the ENCRYPTION attribute of a general tablespace or the mysql system tablespace. The INPLACE algorithm permits concurrent DML on tables that reside in the general tablespace. Concurrent DDL is blocked.

• When a general tablespace or the mysql system tablespace is encrypted, all tables residing in the tablespace are encrypted. Likewise, a table created in an encrypted tablespace is encrypted.

• If the server exits or is stopped during normal operation, it is recommended to restart the server using the same encryption settings that were configured previously.

• The first master encryption key is generated when the first new or existing tablespace is encrypted.

• Master key rotation re-encrypts tablespaces keys but does not change the tablespace key itself. To change a tablespace key, you must disable and re-enable encryption. For file-per-table tablespaces, re-encrypting the tablespace is an ALGORITHM=COPY operation that rebuilds the table. For general tablespaces and the mysql system tablespace, it is an ALGORITHM=INPLACE operation, which does not require rebuilding tables that reside in the tablespace.

• If a table is created with both the COMPRESSION and ENCRYPTION options, compression is performed before tablespace data is encrypted.

• If a keyring data file (the file named by keyring\_file\_data or keyring\_encrypted\_file\_data) is empty or missing, the first execution of ALTER INSTANCE ROTATE INNODB MASTER KEY creates a master encryption key.

• Uninstalling the component\_keyring\_file or component\_keyring\_encrypted\_file component does not remove an existing keyring data file. Uninstalling the keyring\_file or keyring\_encrypted\_file plugin does not remove an existing keyring data file.

• It is recommended that you not place a keyring data file under the same directory as tablespace data files.

• Modifying the keyring\_file\_data or keyring\_encrypted\_file\_data setting at runtime or when restarting the server can cause previously encrypted tablespaces to become inaccessible, resulting in lost data.

• Encryption is supported for the InnoDB FULLTEXT index tables that are created implicitly when adding a FULLTEXT index. For related information, see InnoDB Full-Text Index Tables.

**Encryption** **Limitations**

• Advanced Encryption Standard (AES) is the only supported encryption algorithm. InnoDB tablespace encryption uses Electronic Codebook (ECB) block encryption mode for tablespace key encryption and Cipher Block Chaining (CBC) block encryption mode for data encryption. Padding is not used with CBC block encryption mode. Instead, InnoDB ensures that the text to be encrypted is a multiple of the block size.

• Encryption is only supported for file-per-table tablespaces, general tablespaces, and the mysql system tablespace. Encryption support for general tablespaces was introduced in MySQL 8.0.13. Encryption support for the mysql system tablespace is available as of MySQL 8.0.16. Encryption is not supported for other tablespace types including the InnoDB system tablespace.

• You cannot move or copy a table from an encrypted file-per-table tablespace, general tablespace, or the mysql system tablespace to a tablespace type that does not support encryption.

• You cannot move or copy a table from an encrypted tablespace to an unencrypted tablespace. However, moving a table from an unencrypted tablespace to an encrypted one is permitted. For example, you can move or copy a table from a unencrypted file-per-table or general tablespace to an

encrypted general tablespace.

• By default, tablespace encryption only applies to data in the tablespace. Redo log and undo log data can be encrypted by enabling [innodb\_redo\_log\_encrypt](#_bookmark95) and [innodb\_undo\_log\_encrypt](#_bookmark96). See [Redo Log Encryption](#_bookmark85), and [Undo Log Encryption](#_bookmark86). For information about binary log file and relay log file encryption, see Section 17.3.2, “Encrypting Binary Log Files and Relay Log Files” .

• It is not permitted to change the storage engine of a table that resides in, or previously resided in, an encrypted tablespace.

**15.14** **InnoDB** **Startup** **Options** **and** **System** **Variables**

• System variables that are true or false can be enabled at server startup by naming them, or disabled by using a --skip- prefix. For example, to enable or disable the InnoDB adaptive hash index, you can use [--innodb-adaptive-hash-index](#_bookmark97) or [--skip-innodb-](#_bookmark97) [adaptive-hash-index](#_bookmark97) on the command line, or [innodb\_adaptive\_hash\_index](#_bookmark97) or skip\_innodb\_adaptive\_hash\_index in an option file.

• Some variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the SET statement by setting them to ON or 1, or disabled by setting them to OFF or 0. Boolean variables can be set at startup to the values ON, TRUE, OFF, and FALSE (not case- sensitive), as well as 1 and 0. See Section 4.2.2.4, “Program Option Modifiers” .

• System variables that take a numeric value can be specified as --*var\_name*=*value* on the command line or as *var\_name*=*value* in option files.

• Many system variables can be changed at runtime (see Section 5.1.9.2, “Dynamic System Variables” ).

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

• Certain options control the locations and layout of the InnoDB data files. Section 15.8.1, “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** **15.24** **InnoDB** **Option** **and** **Variable** **Reference**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| daemon\_mem | aecshed\_enable | Y\_sinlog | Yes |  | Global | No |
| daemon\_mem | aecshed\_engine | Y\_eli\_name | Yes |  | Global | No |
| daemon\_mem | aecshed\_engine | Y\_eli\_path | Yes |  | Global | No |
| daemon\_mem | aecshed\_option | Yes | Yes |  | Global | No |
| daemon\_mem | aecshed\_r\_batc | \_ze | Yes |  | Global | No |
| daemon\_mem | aecshed\_w\_bat | cYhessize\_ | Yes |  | Global | No |
| foreign\_key\_c | hecks |  | Yes |  | Both | Yes |
| [innodb](#_bookmark98) | Yes | Yes |  |  |  |  |
| innodb\_adapti | vYee\_lushing | Yes | Yes |  | Global | Yes |
| innodb\_adapti | vYee\_lushing\_lw | Yes  m | Yes |  | Global | Yes |
| innodb\_adapti | vYeeshash\_\_index | Yes | Yes |  | Global | Yes |
| innodb\_adapti | vYeeshash\_\_index | Y\_peasrts | Yes |  | Global | No |
| innodb\_adapti | vYeesmax\_\_sleep\_ | Ydeslay | Yes |  | Global | Yes |
| innodb\_api\_bk | Y\_ceosmmit\_inter | vYaels | Yes |  | Global | Yes |
| innodb\_api\_di | sYasle\_rowlock | Yes | Yes |  | Global | No |
| innodb\_api\_e | nlse\_binlog | Yes | Yes |  | Global | No |
| innodb\_api\_e | nlse\_mdl | Yes | Yes |  | Global | No |
| innodb\_api\_trx | Y\_leeel | Yes | Yes |  | Global | Yes |
| innodb\_autoe | xYteensd\_increment | Yes | Yes |  | Global | Yes |
| innodb\_autoin | cY\_elosck\_mode | Yes | Yes |  | Global | No |
| innodb\_backg | rd\_drop\_lis | tY\_epty | Yes |  | Global | Yes |
| Innodb\_buffer\_ | pool\_bytes\_d | ata |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_bytes\_di | rty |  | Yes | Global | No |
| innodb\_buffer\_ | l\_chunk\_s | ies | Yes |  | Global | No |
| innodb\_buffer\_ | l\_debug | Yes | Yes |  | Global | No |
| innodb\_buffer\_ | l\_dump\_at | \_utdown | Yes |  | Global | Yes |
| innodb\_buffer\_ | l\_dump\_n | oYwes | Yes |  | Global | Yes |
| innodb\_buffer\_ | l\_dump\_p | cYtes | Yes |  | Global | Yes |
| Innodb\_buffer\_ | pool\_dump\_s | tatus |  | Yes | Global | No |
| innodb\_buffer\_ | l\_filename | Yes | Yes |  | Global | Yes |
| innodb\_buffer\_ | l\_in\_core\_ | fYiles | Yes |  | Global | Yes |
| innodb\_buffer\_ | l\_instance | Yes  s | Yes |  | Global | No |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| innodb\_buffer\_ | l\_load\_ab | otes | Yes |  | Global | Yes |
| innodb\_buffer\_ | l\_load\_at\_ | ttup | Yes |  | Global | No |
| innodb\_buffer\_ | l\_load\_now | Yes | Yes |  | Global | Yes |
| Innodb\_buffer\_ | pool\_load\_sta | tus |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_d | ata |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_d | irty |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_f | lushed |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_f | ree |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_l | atched |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_ | misc |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_pages\_t | otal |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_read\_ah | ead |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_read\_ah | ead\_evicted |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_read\_ah | ead\_rnd |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_read\_re | quests |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_reads |  |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_resize\_s | tatus |  | Yes | Global | No |
| innodb\_buffer\_ | l\_size | Yes | Yes |  | Global | Yes |
| Innodb\_buffer\_ | pool\_wait\_free |  |  | Yes | Global | No |
| Innodb\_buffer\_ | pool\_write\_re | quests |  | Yes | Global | No |
| innodb\_chang | eYebsu\_ffer\_max\_ | i | Yes |  | Global | Yes |
| innodb\_chang | eYebsu\_ffering | Yes | Yes |  | Global | Yes |
| innodb\_chang | eYebsu\_ffering\_de | uegs | Yes |  | Global | Yes |
| innodb\_check | pYoeinst\_disabled | Yes | Yes |  | Global | Yes |
| innodb\_check | s\_algorithm | Yes | Yes |  | Global | Yes |
| innodb\_cmp\_ | pYeres\_index\_ena | bYl**e**ds | Yes |  | Global | Yes |
| innodb\_comm | itY\_ecncurrency | Yes | Yes |  | Global | Yes |
| innodb\_compr | eYsess\_debug | Yes | Yes |  | Global | Yes |
| innodb\_compr | eYsession\_failure\_ | Ythshold\_pct | Yes |  | Global | Yes |
| innodb\_compr | eYsession\_level | Yes | Yes |  | Global | Yes |
| innodb\_compr | eYsession\_pad\_pct | Yems\_ax | Yes |  | Global | Yes |
| innodb\_concu | r**e**nscy\_tickets | Yes | Yes |  | Global | Yes |
| innodb\_data\_f | iYlee\_spath | Yes | Yes |  | Global | No |
| Innodb\_data\_f | syncs |  |  | Yes | Global | No |
| innodb\_data\_ | hse\_dir | Yes | Yes |  | Global | No |
| Innodb\_data\_ | pending\_fsyncs |  |  | Yes | Global | No |
| Innodb\_data\_ | pending\_reads |  |  | Yes | Global | No |
| Innodb\_data\_ | pending\_writes |  |  | Yes | Global | No |
| Innodb\_data\_ | read |  |  | Yes | Global | No |
| Innodb\_data\_ | reads |  |  | Yes | Global | No |
| Innodb\_data\_ | writes |  |  | Yes | Global | No |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| Innodb\_data\_ | written |  |  | Yes | Global | No |
| Innodb\_dblwr\_ | pages\_written |  |  | Yes | Global | No |
| Innodb\_dblwr\_ | writes |  |  | Yes | Global | No |
| innodb\_ddl\_b | ufers\_size | Yes | Yes |  | Both | Yes |
| innodb\_ddl\_log | Y\_ecash\_reset\_ | eug | Yes |  | Global | Yes |
| innodb\_ddl\_th | rs | Yes | Yes |  | Both | Yes |
| innodb\_deadl | okes\_detect | Yes | Yes |  | Global | Yes |
| innodb\_dedica | tYeeds\_server | Yes | Yes |  | Global | No |
| innodb\_defaul | tY\_osw\_format | Yes | Yes |  | Global | Yes |
| innodb\_direct | oies | Yes | Yes |  | Global | No |
| innodb\_disable | Y\_rt\_file\_cac | **e**s | Yes |  | Global | Yes |
| innodb\_doubl | eie | Yes | Yes |  | Global | Varies |
| innodb\_doubl | eie\_batch\_s | i**e**s | Yes |  | Global | No |
| innodb\_doubl | eie\_dir | Yes | Yes |  | Global | No |
| innodb\_doubl | eie\_files | Yes | Yes |  | Global | No |
| innodb\_doubl | eie\_pages | Yes | Yes |  | Global | No |
| innodb\_fast\_s | hYudsown | Yes | Yes |  | Global | Yes |
| innodb\_fil\_ma | kYee\_age\_dirty\_ | eug | Yes |  | Global | Yes |
| innodb\_file\_pe | Yret\_ble | Yes | Yes |  | Global | Yes |
| innodb\_fill\_fac | tYos | Yes | Yes |  | Global | Yes |
| innodb\_flush\_ | lge\_sat\_timeout | Yes | Yes |  | Global | Yes |
| innodb\_flush\_ | lge\_sat\_trx\_com | eits | Yes |  | Global | Yes |
| innodb\_flush\_ | mYetshod | Yes | Yes |  | Global | No |
| innodb\_flush\_ | nYeeigshbors | Yes | Yes |  | Global | Yes |
| innodb\_flush\_ | s | Yes | Yes |  | Global | Yes |
| innodb\_flushin | \_easvg\_loops | Yes | Yes |  | Global | Yes |
| innodb\_force\_ | l\_corrupted | Yes | Yes |  | Global | No |
| innodb\_force\_ | rYevery | Yes | Yes |  | Global | No |
| innodb\_fsync\_ | tYhesshold | Yes | Yes |  | Global | Yes |
| innodb\_ft\_aux\_ | table |  | Yes |  | Global | Yes |
| innodb\_ft\_cac | hYeessize\_ | Yes | Yes |  | Global | No |
| innodb\_ft\_ena | les\_diag\_print | Yes | Yes |  | Global | Yes |
| innodb\_ft\_ena | les\_stopword | Yes | Yes |  | Both | Yes |
| innodb\_ft\_max | Y\_teosken\_size | Yes | Yes |  | Global | No |
| innodb\_ft\_min\_ | Yten\_size | Yes | Yes |  | Global | No |
| innodb\_ft\_num | Y\_sord\_optimiz | es | Yes |  | Global | Yes |
| innodb\_ft\_res | uYlte\_ache\_limit | Yes | Yes |  | Global | Yes |
| innodb\_ft\_ser | vYeres\_stopword\_t | bel | Yes |  | Global | Yes |
| innodb\_ft\_sort | \_Ypells\_degree | Yes | Yes |  | Global | No |
| innodb\_ft\_tota | lY\_asche\_size | Yes | Yes |  | Global | No |
| innodb\_ft\_use | rY\_tsopword\_ta | bYl**e**s | Yes |  | Both | Yes |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| Innodb\_have\_ | atomic\_builtins |  |  | Yes | Global | No |
| innodb\_idle\_fl | us\_pct | Yes | Yes |  | Global | Yes |
| innodb\_io\_cap | ceiy | Yes | Yes |  | Global | Yes |
| innodb\_io\_cap | ceiy\_max | Yes | Yes |  | Global | Yes |
| innodb\_limit\_o | teiistic\_insert | Y\_bug | Yes |  | Global | Yes |
| innodb\_lock\_w | iets\_timeout | Yes | Yes |  | Both | Yes |
| innodb\_log\_b | ufers\_size | Yes | Yes |  | Global | Varies |
| innodb\_log\_ch | cekspoint\_fuzzy | Y\_w | Yes |  | Global | Yes |
| innodb\_log\_ch | cekspoint\_now | Yes | Yes |  | Global | Yes |
| innodb\_log\_ch | cek**s**ums | Yes | Yes |  | Global | Yes |
| innodb\_log\_co | epessed\_pa | gs | Yes |  | Global | Yes |
| innodb\_log\_file | Y\_size | Yes | Yes |  | Global | No |
| innodb\_log\_fil | ee\_sin\_group | Yes | Yes |  | Global | No |
| innodb\_log\_gr | ueps\_home\_dir | Yes | Yes |  | Global | No |
| innodb\_log\_sp | Yine\_pu\_abs\_lw | es | Yes |  | Global | Yes |
| innodb\_log\_sp | Yine\_pu\_pct\_hw | es | Yes |  | Global | Yes |
| innodb\_log\_w | aYitesfor\_\_flush\_s | pYine\_shwm | Yes |  | Global | Yes |
| Innodb\_log\_w | aits |  |  | Yes | Global | No |
| innodb\_log\_w | r\_ahead\_size | Yes | Yes |  | Global | Yes |
| Innodb\_log\_w | rite\_requests |  |  | Yes | Global | No |
| innodb\_log\_w | rYiters\_threads | Yes | Yes |  | Global | Yes |
| Innodb\_log\_w | rites |  |  | Yes | Global | No |
| innodb\_lru\_sc | aYne\_sdepth | Yes | Yes |  | Global | Yes |
| innodb\_max\_ | dYirteys\_pages\_pct | Yes | Yes |  | Global | Yes |
| innodb\_max\_ | dYirteys\_pages\_pct | Yl\_sm | Yes |  | Global | Yes |
| innodb\_max\_ | pgse\_lag | Yes | Yes |  | Global | Yes |
| innodb\_max\_ | pgse\_lag\_dela | Yes  y | Yes |  | Global | Yes |
| innodb\_max\_ | u\_log\_size | Yes | Yes |  | Global | Yes |
| innodb\_merge | Yt\_eshold\_set | Yaells debug | Yes |  | Global | Yes |
| innodb\_monitor | Yedsi\_sable | Yes | Yes |  | Global | Yes |
| innodb\_monitor | Yeesn\_able | Yes | Yes |  | Global | Yes |
| innodb\_monitor | Yer\_set | Yes | Yes |  | Global | Yes |
| innodb\_monitor | Yer\_set\_all | Yes | Yes |  | Global | Yes |
| Innodb\_num\_ | open\_files |  |  | Yes | Global | No |
| innodb\_numa\_ | Yinrleave | Yes | Yes |  | Global | No |
| innodb\_old\_bl | oYcs\_pct | Yes | Yes |  | Global | Yes |
| innodb\_old\_bl | oYcs\_time | Yes | Yes |  | Global | Yes |
| innodb\_online\_ | Yaeltr\_log\_max | Y\_ie | Yes |  | Global | Yes |
| innodb\_open\_ | fYiles | Yes | Yes |  | Global | Varies |
| innodb\_optimi | zYee\_ulltext\_only | Yes | Yes |  | Global | Yes |
| Innodb\_os\_log | \_fsyncs |  |  | Yes | Global | No |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| Innodb\_os\_log | \_pending\_fsyn | cs |  | Yes | Global | No |
| Innodb\_os\_log | \_pending\_writ | es |  | Yes | Global | No |
| Innodb\_os\_log | \_written |  |  | Yes | Global | No |
| innodb\_page\_ | cYleasners | Yes | Yes |  | Global | No |
| Innodb\_page\_ | size |  |  | Yes | Global | No |
| innodb\_page\_ | sYizees | Yes | Yes |  | Global | No |
| Innodb\_pages | \_created |  |  | Yes | Global | No |
| Innodb\_pages | \_read |  |  | Yes | Global | No |
| Innodb\_pages | \_written |  |  | Yes | Global | No |
| innodb\_parallel | Y\_ad\_threads | Yes | Yes |  | Session | Yes |
| innodb\_print\_ | aYlle\_dseadlocks | Yes | Yes |  | Global | Yes |
| innodb\_print\_ | dles\_logs | Yes | Yes |  | Global | Yes |
| innodb\_purge\_ | ch\_size | Yes | Yes |  | Global | Yes |
| innodb\_purge\_ | s**e**sg\_truncate\_ | Yfreesquency | Yes |  | Global | Yes |
| innodb\_purge\_ | Ytherads | Yes | Yes |  | Global | No |
| innodb\_random | Yer\_ad\_ahead | Yes | Yes |  | Global | Yes |
| innodb\_read\_ | aed\_thresho | les | Yes |  | Global | Yes |
| innodb\_read\_i | oYet\_reads | Yes | Yes |  | Global | No |
| innodb\_read\_ | oelys | Yes | Yes |  | Global | No |
| innodb\_redo\_l | oYge\_sarchive\_dir | Yes  s | Yes |  | Global | Yes |
| innodb\_redo\_l | ge\_scapacity | Yes | Yes |  | Global | Yes |
| Innodb\_redo\_ | log\_capacity\_r | esized |  | Yes | Global | No |
| Innodb\_redo\_ | log\_checkpoint\_ | lsn |  | Yes | Global | No |
| Innodb\_redo\_ | log\_current\_lsn |  |  | Yes | Global | No |
| Innodb\_redo\_ | log\_enabled |  |  | Yes | Global | No |
| innodb\_redo\_l | ge\_sencrypt | Yes | Yes |  | Global | Yes |
| Innodb\_redo\_ | log\_flushed\_to\_ | disk\_lsn |  | Yes | Global | No |
| Innodb\_redo\_ | log\_logical\_size |  |  | Yes | Global | No |
| Innodb\_redo\_ | log\_physical\_s | ize |  | Yes | Global | No |
| Innodb\_redo\_ | log\_read\_only |  |  | Yes | Global | No |
| Innodb\_redo\_ | log\_resize\_stat | us |  | Yes | Global | No |
| Innodb\_redo\_ | log\_uuid |  |  | Yes | Global | No |
| innodb\_replica | tYioe\_delay | Yes | Yes |  | Global | Yes |
| innodb\_rollbac | \_n\_timeout | Yes | Yes |  | Global | No |
| innodb\_rollbac | Ykesse\_gments | Yes | Yes |  | Global | Yes |
| Innodb\_row\_l | ock\_current\_wa | its |  | Yes | Global | No |
| Innodb\_row\_l | ock\_time |  |  | Yes | Global | No |
| Innodb\_row\_l | ock\_time\_avg |  |  | Yes | Global | No |
| Innodb\_row\_l | ock\_time\_max |  |  | Yes | Global | No |
| Innodb\_row\_l | ock\_waits |  |  | Yes | Global | No |
| Innodb\_rows\_ | deleted |  |  | Yes | Global | No |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| Innodb\_rows\_ | inserted |  |  | Yes | Global | No |
| Innodb\_rows\_ | read |  |  | Yes | Global | No |
| Innodb\_rows\_ | updated |  |  | Yes | Global | No |
| innodb\_saved\_ | Ypeasge\_number | \_bug | Yes |  | Global | Yes |
| innodb\_segm | eYntesreserve\_\_fa | cYtoers | Yes |  | Global | Yes |
| innodb\_sort\_b | uYfesr\_size | Yes | Yes |  | Global | No |
| innodb\_spin\_ | weits\_delay | Yes | Yes |  | Global | Yes |
| innodb\_spin\_ | weits\_pause\_mu | lYtilsier | Yes |  | Global | Yes |
| innodb\_stats\_ | aYuos\_recalc | Yes | Yes |  | Global | Yes |
| innodb\_stats\_ | i celde\_delete\_ | Ymeasrked | Yes |  | Global | Yes |
| innodb\_stats\_ | mYetshod | Yes | Yes |  | Global | Yes |
| innodb\_stats\_ | oYnesmetadata\_ | Yes | Yes |  | Global | Yes |
| innodb\_stats\_ | pYesistent | Yes | Yes |  | Global | Yes |
| innodb\_stats\_ | pYesistent\_sam | les\_pages | Yes |  | Global | Yes |
| innodb\_stats\_ | taensient\_samp | Ylee\_spages | Yes |  | Global | Yes |
| [innodb](#_bookmark99)-  [status-file](#_bookmark99) | Yes | Yes |  |  |  |  |
| innodb\_status\_ | Yoeuput | Yes | Yes |  | Global | Yes |
| innodb\_status\_ | Yoeuput\_locks | Yes | Yes |  | Global | Yes |
| innodb\_strict\_ | mYoedse | Yes | Yes |  | Both | Yes |
| innodb\_sync\_ | areasy\_size | Yes | Yes |  | Global | No |
| innodb\_sync\_ | dYeg | Yes | Yes |  | Global | No |
| innodb\_sync\_ | s\_loops | Yes | Yes |  | Global | Yes |
| Innodb\_system | \_rows\_deleted |  |  | Yes | Global | No |
| Innodb\_system | \_rows\_inserte | d |  | Yes | Global | No |
| Innodb\_system | \_rows\_read |  |  | Yes | Global | No |
| innodb\_table\_ | lcek**s** | Yes | Yes |  | Both | Yes |
| innodb\_temp\_ | dYa\_file\_path | Yes | Yes |  | Global | No |
| innodb\_temp\_ | tslespaces\_d | iYres | Yes |  | Global | No |
| innodb\_thread | \_currency | Yes | Yes |  | Global | Yes |
| innodb\_thread | Ysel\_ep\_delay | Yes | Yes |  | Global | Yes |
| [innodb\_tmpdir](#_bookmark72) | Yes | Yes | Yes |  | Both | Yes |
| Innodb\_trunca | ted\_status\_wri | tes |  | Yes | Global | No |
| innodb\_trx\_pu | rYges\_view\_upd | aYtees\_only\_debug | Yes |  | Global | Yes |
| innodb\_trx\_rs | ee\_\_slots\_de | bs | Yes |  | Global | Yes |
| innodb\_undo\_ | dYiesctory | Yes | Yes |  | Global | No |
| innodb\_undo\_ | ls\_encrypt | Yes | Yes |  | Global | Yes |
| innodb\_undo\_ | ls\_truncate | Yes | Yes |  | Global | Yes |
| innodb\_undo\_ | tslespaces | Yes | Yes |  | Global | Varies |
| Innodb\_undo\_ | tablespaces\_a | ctive |  | Yes | Global | No |
| Innodb\_undo\_ | tablespaces\_e | xplicit |  | Yes | Global | No |

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Name** | **Cmd-Line** | **Option** **File** | **System** **Var** | **Status** **Var** | **Var** **Scope** | **Dynamic** |
| Innodb\_undo\_ | tablespaces\_i | mplicit |  | Yes | Global | No |
| Innodb\_undo\_ | tablespaces\_t | otal |  | Yes | Global | No |
| innodb\_use\_f | dteasync | Yes | Yes |  | Global | Yes |
| innodb\_use\_n | aYtiev\_aio | Yes | Yes |  | Global | No |
| innodb\_valida | teYet\_blespace\_ | aets | Yes |  | Global | No |
| [innodb\_version](#_bookmark100) |  |  | Yes |  | Global | No |
| innodb\_write\_ | ioYet\_reads | Yes | Yes |  | Global | No |
| unique\_checks |  |  | Yes |  | Both | Yes |

**InnoDB** **Command** **Options**

• [--innodb[=*value*]](#_bookmark98)

|  |  |
| --- | --- |
| 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 5.6.1, “Installing and Uninstalling Plugins” .

To disable InnoDB, use [--innodb=OFF](#_bookmark98) or [--skip-innodb](#_bookmark98). 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](#_bookmark98) and [--skip-](#_bookmark98) [innodb](#_bookmark98) options are deprecated and have no effect. Their use results in a warning. Expect these options to be removed in a future MySQL release.

• [--innodb-status-file](#_bookmark99)

|  |  |
| --- | --- |
| 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 15.17.2, “Enabling InnoDB Monitors”](#_bookmark101) .

• [--skip-innodb](#_bookmark98)

Disable the InnoDB storage engine. See the description of [--innodb](#_bookmark98).

**InnoDB** **System** **Variables**

• [daemon\_memcached\_enable\_binlog](#_bookmark102)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-enable-  binlog[={OFF |ON}] |
| System Variable | [daemon\_memcached\_enable\_binlog](#_bookmark102) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.20.7, “The InnoDB memcached Plugin and Replication” .

• [daemon\_memcached\_engine\_lib\_name](#_bookmark103)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-engine-lib-  name=file\_name |
| System Variable | [daemon\_memcached\_engine\_lib\_name](#_bookmark103) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | File name |
| Default Value | innodb\_engine.so |

Specifies the shared library that implements the InnoDB memcached plugin. For more information, see Section 15.20.3, “Setting Up the InnoDB memcached Plugin” .

• [daemon\_memcached\_engine\_lib\_path](#_bookmark104)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-engine-lib-  path=dir\_name |
| System Variable | [daemon\_memcached\_engine\_lib\_path](#_bookmark104) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.20.3, “Setting Up the InnoDB memcached Plugin” .

• [daemon\_memcached\_option](#_bookmark106)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-option=options |
| System Variable | [daemon\_memcached\_option](#_bookmark106) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.20.3, “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](#_bookmark107)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-r-batch-size=# |
| System Variable | [daemon\_memcached\_r\_batch\_size](#_bookmark107) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark105).

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 15.20.3, “Setting Up the InnoDB memcached Plugin” .

• [daemon\_memcached\_w\_batch\_size](#_bookmark105)

|  |  |
| --- | --- |
| Command-Line Format | --daemon-memcached-w-batch-size=# |

|  |  |
| --- | --- |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark107).

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 15.20.3, “Setting Up the InnoDB memcached Plugin” .

• [innodb\_adaptive\_flushing](#_bookmark108)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-adaptive-flushing[={OFF |  ON}] |
| System Variable | [innodb\_adaptive\_flushing](#_bookmark108) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Specifies whether to dynamically adjust the rate of flushing dirty pages in the InnoDB buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Section 15.8.3.5, “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](#_bookmark109)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-adaptive-flushing-lwm=# |
| System Variable | [innodb\_adaptive\_flushing\_lwm](#_bookmark109) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 10 |
| Minimum Value | 0 |
| Maximum Value | 70 |

Defines the low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing” .

• [innodb\_adaptive\_hash\_index](#_bookmark97)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-adaptive-hash-index[={OFF |  ON}] |
| System Variable | [innodb\_adaptive\_hash\_index](#_bookmark97) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 15.5.3, “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 5.1.9.1, “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](#_bookmark110)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-adaptive-hash-index-parts=# |
| System Variable | [innodb\_adaptive\_hash\_index\_parts](#_bookmark110) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Numeric |
| Default Value | 8 |
| Minimum Value | 1 |
| Maximum Value | 512 |

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

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

512.

For related information, see Section 15.5.3, “Adaptive Hash Index” .

• [innodb\_adaptive\_max\_sleep\_delay](#_bookmark111)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-adaptive-max-sleep-delay=# |
| System Variable | [innodb\_adaptive\_max\_sleep\_delay](#_bookmark111) |

|  |  |
| --- | --- |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 150000 |
| Minimum Value | 0 |
| Maximum Value | 1000000 |
| Unit | microseconds |

Permits InnoDB to automatically adjust the value of [innodb\_thread\_sleep\_delay](#_bookmark112) up

or down according to the current workload. Any nonzero value enables automated, dynamic adjustment of the [innodb\_thread\_sleep\_delay](#_bookmark112) value, up to the maximum value specified in the [innodb\_adaptive\_max\_sleep\_delay](#_bookmark111) 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 15.8.4, “Configuring Thread Concurrency for InnoDB” .

• [innodb\_api\_bk\_commit\_interval](#_bookmark113)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-api-bk-commit-interval=# |
| System Variable | [innodb\_api\_bk\_commit\_interval](#_bookmark113) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 15.20.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin” .

• [innodb\_api\_disable\_rowlock](#_bookmark114)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-api-disable-rowlock[={OFF |  ON}] |
| System Variable | [innodb\_api\_disable\_rowlock](#_bookmark114) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark114) is disabled, which means that memcached requests row locks for

get and set operations. When [innodb\_api\_disable\_rowlock](#_bookmark114) is enabled, memcached requests a table lock instead of row locks.

[innodb\_api\_disable\_rowlock](#_bookmark114) 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 15.20.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin” .

• [innodb\_api\_enable\_binlog](#_bookmark115)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-api-enable-binlog[={OFF |  ON}] |
| System Variable | [innodb\_api\_enable\_binlog](#_bookmark115) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark116)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-api-enable-mdl[={OFF |ON}] |
| System Variable | [innodb\_api\_enable\_mdl](#_bookmark116) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.20.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin” .

• [innodb\_api\_trx\_level](#_bookmark117)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-api-trx-level=# |
| System Variable | [innodb\_api\_trx\_level](#_bookmark117) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 15.20.6.4, “Controlling Transactional Behavior of the InnoDB memcached Plugin” .

• [innodb\_autoextend\_increment](#_bookmark118)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-autoextend-increment=# |
| System Variable | [innodb\_autoextend\_increment](#_bookmark118) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 64 |
| Minimum Value | 1 |
| Maximum Value | 1000 |
| Unit | megabytes |

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

The [innodb\_autoextend\_increment](#_bookmark118) setting does not affect file-per-table tablespace

files or general tablespace files. These files are auto-extending regardless of the [innodb\_autoextend\_increment](#_bookmark118) setting. The initial extensions are by small amounts, after which extensions occur in increments of 4MB.

• [innodb\_autoinc\_lock\_mode](#_bookmark119)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-autoinc-lock-mode=# |
| System Variable | [innodb\_autoinc\_lock\_mode](#_bookmark119) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 2 |
| 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 2 (interleaved) as of MySQL 8.0, and 1 (consecutive) before that. The change to interleaved lock mode as the default setting reflects the change from statement-based to row- based replication as the default replication type, which occurred in MySQL 5.7. Statement-based replication requires the consecutive auto-increment lock mode to ensure that auto-increment values are assigned in a predictable and repeatable order for a given sequence of SQL statements, whereas row-based replication is not sensitive to the execution order of SQL statements.

For the characteristics of each lock mode, see InnoDB AUTO\_INCREMENT Lock Modes.

• [innodb\_background\_drop\_list\_empty](#_bookmark120)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-background-drop-list-  empty[={OFF |ON}] |
| System Variable | [innodb\_background\_drop\_list\_empty](#_bookmark120) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enabling the [innodb\_background\_drop\_list\_empty](#_bookmark120) 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](#_bookmark121)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-chunk-size=# |
| System Variable | [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 134217728 |
| Minimum Value | 1048576 |
| Maximum Value | innodb\_buffer\_pool\_size /  innodb\_buffer\_pool\_instances |
| Unit | bytes |

[innodb\_buffer\_pool\_chunk\_size](#_bookmark121) 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](#_bookmark121) is 128MB (134217728 bytes). The number of pages contained in a chunk depends on the value of [innodb\_page\_size](#_bookmark7).



[innodb\_buffer\_pool\_chunk\_size](#_bookmark121) can be increased or decreased in units of 1MB (1048576 bytes).

The following conditions apply when altering the [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) value:

• If [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122) is larger than the current buffer pool size when the buffer pool is initialized, [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) is truncated to [innodb\_buffer\_pool\_size](#_bookmark14) / [innodb\_buffer\_pool\_instances](#_bookmark122).

• Buffer pool size must always be equal to or a multiple of [innodb\_buffer\_pool\_chunk\_size](#_bookmark121)

\* [innodb\_buffer\_pool\_instances](#_bookmark122). If you alter [innodb\_buffer\_pool\_chunk\_size](#_bookmark121), [innodb\_buffer\_pool\_size](#_bookmark14) is automatically rounded to a value that is equal to or a multiple of [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122). The adjustment occurs when the buffer pool is initialized.

**Important**

Care should be taken when changing [innodb\_buffer\_pool\_chunk\_size](#_bookmark121), as changing this value can automatically increase the size of the buffer pool. Before changing [innodb\_buffer\_pool\_chunk\_size](#_bookmark121), calculate its effect on [innodb\_buffer\_pool\_size](#_bookmark14) to ensure that the resulting buffer pool size is acceptable.

To avoid potential performance issues, the number of chunks ([innodb\_buffer\_pool\_size](#_bookmark14) / [innodb\_buffer\_pool\_chunk\_size](#_bookmark121)) should not exceed 1000.

The [innodb\_buffer\_pool\_size](#_bookmark14) 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](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122), and changing either of those variable settings requires restarting the server.

See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

• [innodb\_buffer\_pool\_debug](#_bookmark123)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-debug[={OFF |  ON}] |
| System Variable | [innodb\_buffer\_pool\_debug](#_bookmark123) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enabling this option permits multiple buffer pool instances when the buffer pool is less

than 1GB in size, ignoring the 1GB minimum buffer pool size constraint imposed on [innodb\_buffer\_pool\_instances](#_bookmark122). The [innodb\_buffer\_pool\_debug](#_bookmark123) option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

• [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-dump-at-  shutdown[={OFF |ON}] |
| System Variable | [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124) |

|  |  |
| --- | --- |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Specifies whether to record the pages cached in the InnoDB buffer pool when the MySQL server is shut down, to shorten the warmup process at the next restart. Typically used in combination with [innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125). The [innodb\_buffer\_pool\_dump\_pct](#_bookmark126) option defines the percentage of most recently used buffer pool pages to dump.

Both [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124) and

[innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125) are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_dump\_now](#_bookmark127)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-dump-now[={OFF |  ON}] |
| System Variable | [innodb\_buffer\_pool\_dump\_now](#_bookmark127) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Immediately makes a record of pages cached in the InnoDB buffer pool. Typically used in combination with [innodb\_buffer\_pool\_load\_now](#_bookmark128).

Enabling [innodb\_buffer\_pool\_dump\_now](#_bookmark127) triggers the recording action but does not alter the variable setting, which always remains OFF or 0. To view buffer pool dump status after triggering a dump, query the Innodb\_buffer\_pool\_dump\_status variable.

Enabling [innodb\_buffer\_pool\_dump\_now](#_bookmark127) triggers the dump action but does not alter the variable setting, which always remains OFF or 0. To view buffer pool dump status after triggering a dump, query the Innodb\_buffer\_pool\_dump\_status variable.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_dump\_pct](#_bookmark126)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-dump-pct=# |
| System Variable | [innodb\_buffer\_pool\_dump\_pct](#_bookmark126) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 25 |
| Minimum Value | 1 |

|  |  |
| --- | --- |
| Maximum Value | 100 |

Specifies the percentage of the most recently used pages for each buffer pool to read out and dump. The range is 1 to 100. The default value is 25. For example, if there are 4 buffer pools with 100 pages each, and [innodb\_buffer\_pool\_dump\_pct](#_bookmark126) is set to 25, the 25 most recently used pages from each buffer pool are dumped.

• [innodb\_buffer\_pool\_filename](#_bookmark129)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-  filename=file\_name |
| System Variable | [innodb\_buffer\_pool\_filename](#_bookmark129) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | File name |
| Default Value | ib\_buffer\_pool |

Specifies the name of the file that holds the list of tablespace IDs and page IDs produced by [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124) or [innodb\_buffer\_pool\_dump\_now](#_bookmark127). 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 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_in\_core\_file](#_bookmark130)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-in-core-  file[={OFF |ON}] |
| Introduced | 8.0.14 |
| System Variable | [innodb\_buffer\_pool\_in\_core\_file](#_bookmark130) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Disabling the [innodb\_buffer\_pool\_in\_core\_file](#_bookmark130) variable reduces the size of core files by excluding InnoDB buffer pool pages. To use this variable, the core\_file variable must be enabled and the operating system must support the MADV\_DONTDUMP non-POSIX extension to madvise(), which is supported in Linux 3.4 and later. For more information, see Section 15.8.3.7, “Excluding Buffer Pool Pages from Core Files” .

• [innodb\_buffer\_pool\_instances](#_bookmark122)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-instances=# |
| System Variable | [innodb\_buffer\_pool\_instances](#_bookmark122) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark14) 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](#_bookmark122) and [innodb\_buffer\_pool\_size](#_bookmark14) 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](#_bookmark14), as described below:

• If [innodb\_buffer\_pool\_size](#_bookmark14) is greater than 1.3GB, the default for [innodb\_buffer\_pool\_instances](#_bookmark122) is [innodb\_buffer\_pool\_size](#_bookmark14)/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](#_bookmark14) is greater than or equal to 1GB. Otherwise, the default is 1.

For related information, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” .

• [innodb\_buffer\_pool\_load\_abort](#_bookmark131)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-load-  abort[={OFF |ON}] |
| System Variable | [innodb\_buffer\_pool\_load\_abort](#_bookmark131) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |

|  |  |
| --- | --- |
| Default Value | OFF |

Interrupts the process of restoring InnoDB buffer pool contents triggered by [innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125) or [innodb\_buffer\_pool\_load\_now](#_bookmark128).

Enabling [innodb\_buffer\_pool\_load\_abort](#_bookmark131) 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 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-load-at-  startup[={OFF |ON}] |
| System Variable | [innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Specifies that, on MySQL server startup, the InnoDB buffer pool is automatically warmed up by loading the same pages it held at an earlier time. Typically used in combination with [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124).

Both [innodb\_buffer\_pool\_dump\_at\_shutdown](#_bookmark124) and

[innodb\_buffer\_pool\_load\_at\_startup](#_bookmark125) are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_load\_now](#_bookmark128)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-load-now[={OFF |  ON}] |
| System Variable | [innodb\_buffer\_pool\_load\_now](#_bookmark128) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

Enabling [innodb\_buffer\_pool\_load\_now](#_bookmark128) 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 15.8.3.6, “Saving and Restoring the Buffer Pool State” .

• [innodb\_buffer\_pool\_size](#_bookmark14)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-buffer-pool-size=# |

|  |  |
| --- | --- |
| System Variable | [innodb\_buffer\_pool\_size](#_bookmark14) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 134217728 |
| Minimum Value | 5242880 |
| Maximum Value (64-bit platforms) | 2\*\*64-1 |
| Maximum Value (32-bit platforms) | 2\*\*32-1 |
| Unit | bytes |

The size in bytes of the buffer pool, the memory area where InnoDB caches table and index data. The default value is 134217728 bytes (128MB). The maximum value depends on the CPU architecture; the maximum is 4294967295 (232-1) on 32-bit systems and 18446744073709551615 (264-1) 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](#_bookmark122) 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 15.8.3.6, “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](#_bookmark121) 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](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122). If you alter the buffer pool size to a value that is not equal to or a multiple of [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122), buffer pool size is automatically adjusted to a value that is equal to or a multiple of [innodb\_buffer\_pool\_chunk\_size](#_bookmark121) \* [innodb\_buffer\_pool\_instances](#_bookmark122).

[innodb\_buffer\_pool\_size](#_bookmark14) 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 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

If [innodb\_dedicated\_server](#_bookmark13) is enabled, the [innodb\_buffer\_pool\_size](#_bookmark14) value is automatically configured if it is not explicitly defined. For more information, see [Section 15.8.12,](#_bookmark9) [“Enabling Automatic Configuration for a Dedicated MySQL Server”](#_bookmark9) .

• [innodb\_change\_buffer\_max\_size](#_bookmark132)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-change-buffer-max-size=# |
| System Variable | [innodb\_change\_buffer\_max\_size](#_bookmark132) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 25 |
| Minimum Value | 0 |
| Maximum Value | 50 |

Maximum size for the InnoDB change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For more information, see Section 15.5.2, “Change Buffer” . For general I/O tuning advice, see Section 8.5.8, “Optimizing

InnoDB Disk I/O” .

• [innodb\_change\_buffering](#_bookmark133)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-change-buffering=value |
| System Variable | [innodb\_change\_buffering](#_bookmark133) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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. Values may also be specified numerically.

**Table** **15.25** **Permitted** **Values** **for** **innodb\_change\_buffering**

|  |  |  |
| --- | --- | --- |
| **Value** | **Numeric** **Value** | **Description** |
| none | 0 | Do not buffer any operations. |
| inserts | 1 | Buffer insert operations. |
| deletes | 2 | Buffer delete marking operations; strictly speaking, the writes that mark index records |

•

•

•

|  |  |  |
| --- | --- | --- |
| **Value** | **Numeric** **Value** | **Description** |
|  |  | for later deletion during a purge operation. |
| changes | 3 | Buffer inserts and delete-  marking operations. |
| purges | 4 | Buffer the physical deletion operations that happen in the background. |
| all | 5 | The default. Buffer inserts, delete-marking operations, and purges. |

For more information, see Section 15.5.2, “Change Buffer” . For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O” .

[innodb\_change\_buffering\_debug](#_bookmark134)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-change-buffering-debug=# |
| System Variable | [innodb\_change\_buffering\_debug](#_bookmark134) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 2 |

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

[innodb\_checkpoint\_disabled](#_bookmark135)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-checkpoint-disabled[={OFF |  ON}] |
| System Variable | [innodb\_checkpoint\_disabled](#_bookmark135) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

[innodb\_checksum\_algorithm](#_bookmark136)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-checksum-algorithm=value |

|  |  |
| --- | --- |
| System Variable | [innodb\_checksum\_algorithm](#_bookmark136) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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. The default value for [innodb\_checksum\_algorithm](#_bookmark136) is crc32.

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.

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

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

The following table shows the difference between the none, innodb, and crc32 option values, and their strict counterparts. none, innodb, and crc32 write the specified type of checksum value into each data block, but for compatibility accept other checksum values when verifying a block during a read operation. Strict settings also accept valid checksum values but print an error message when a valid non-matching checksum value is encountered. Using the strict form can make verification faster if all InnoDB data files in an instance are created under an identical [innodb\_checksum\_algorithm](#_bookmark136) value.

**Table** **15.26** **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. |

•

•

|  |  |  |
| --- | --- | --- |
| **Value** | **Generated** **checksum** **(when** **writing)** | **Permitted** **checksums** **(when** **reading)** |
| innodb | A checksum calculated in software, using the original algorithm from InnoDB. | Any of the checksums generated by none, innodb, or  crc32. |
| crc32 | A checksum calculated using the crc32 algorithm, possibly done with a hardware assist. | Any of the checksums generated by none, innodb, or  crc32. |
| strict\_none | A constant number | Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but  non-matching checksum is encountered. |
| strict\_innodb | A checksum calculated in software, using the original algorithm from InnoDB. | Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but  non-matching checksum is encountered. |
| strict\_crc32 | A checksum calculated using the crc32 algorithm, possibly done with a hardware assist. | Any of the checksums generated by none, innodb, or crc32. InnoDB prints an error message if a valid but  non-matching checksum is encountered. |

[innodb\_cmp\_per\_index\_enabled](#_bookmark24)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-cmp-per-index-  enabled[={OFF |ON}] |
| System Variable | [innodb\_cmp\_per\_index\_enabled](#_bookmark24) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

For more information, see Section 26.4.8, “The INFORMATION\_SCHEMA INNODB\_CMP\_PER\_INDEX and INNODB\_CMP\_PER\_INDEX\_RESET Tables” , and [Section 15.9.1.4, “Monitoring InnoDB Table Compression at Runtime”](#_bookmark22) .

[innodb\_commit\_concurrency](#_bookmark137)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-commit-concurrency=# |
| System Variable | [innodb\_commit\_concurrency](#_bookmark137) |
| Scope | Global |
| Dynamic | Yes |

|  |  |
| --- | --- |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 1000 |

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

The value of [innodb\_commit\_concurrency](#_bookmark137) 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](#_bookmark138)

•

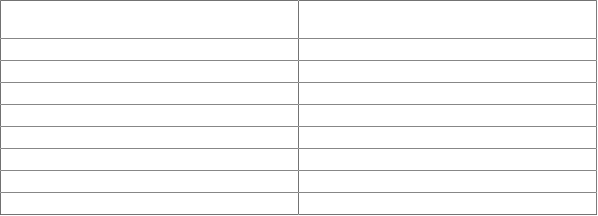
|  |  |
| --- | --- |
| Command-Line Format | --innodb-compress-debug=value |
| System Variable | [innodb\_compress\_debug](#_bookmark138) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Enumeration |
| Default Value | none |
| Valid Values | none  zlib  lz4  lz4hc |

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 15.9.2, “InnoDB Page Compression”](#_bookmark34) .

[innodb\_compression\_failure\_threshold\_pct](#_bookmark26)

•



--innodb-compression-failure-

threshold-pct=#

[innodb\_compression\_failure\_threshold\_pct](#_bookmark26)

Global

Yes

No

Integer

5

0

100

Command-Line Format

System Variable

Scope

Dynamic

SET\_VAR Hint Applies Type

Default Value

Minimum Value

Maximum Value

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](#_bookmark27). A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

For more information, see [Section 15.9.1.6, “Compression for OLTP Workloads”](#_bookmark28) .

• [innodb\_compression\_level](#_bookmark25)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-compression-level=# |
| System Variable | [innodb\_compression\_level](#_bookmark25) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 6 |
| Minimum Value | 0 |
| Maximum Value | 9 |

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

For more information, see [Section 15.9.1.6, “Compression for OLTP Workloads”](#_bookmark28) .

• [innodb\_compression\_pad\_pct\_max](#_bookmark27)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-compression-pad-pct-max=# |
| System Variable | [innodb\_compression\_pad\_pct\_max](#_bookmark27) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50 |
| Minimum Value | 0 |
| Maximum Value | 75 |

Specifies the maximum percentage that can be reserved as free space within each compressed page, allowing room to reorganize the data and modification log within the page when a compressed table or index is updated and the data might be recompressed. Only applies when [innodb\_compression\_failure\_threshold\_pct](#_bookmark26) is set to a nonzero value, and the rate of compression failures passes the cutoff point.

For more information, see [Section 15.9.1.6, “Compression for OLTP Workloads”](#_bookmark28) .

• [innodb\_concurrency\_tickets](#_bookmark139)

|  |  |  |
| --- | --- | --- |
|  | Command-Line Format | --innodb-concurrency-tickets=# |
| System Variable | [innodb\_concurrency\_tickets](#_bookmark139) |
| Scope | Global |
| Dynamic | Yes |
| 3266 | SET\_VAR Hint Applies | No |

|  |  |
| --- | --- |
| Type | Integer |
| Default Value | 5000 |
| Minimum Value | 1 |
| Maximum Value | 4294967295 |

Determines the number of threads that can enter InnoDB concurrently. A thread is placed in a queue when it tries to enter InnoDB if the number of threads has already reached the concurrency limit. When a thread is permitted to enter InnoDB, it is given a number of “ tickets” equal to the value of [innodb\_concurrency\_tickets](#_bookmark139), 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](#_bookmark139) 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](#_bookmark139) 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](#_bookmark139) value, large transactions spend less time waiting for a position at the end of the queue (controlled by [innodb\_thread\_concurrency](#_bookmark140)) 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](#_bookmark139) 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](#_bookmark140) value, you may need to adjust the [innodb\_concurrency\_tickets](#_bookmark139) 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 15.8.4, “Configuring Thread Concurrency for InnoDB” .

• [innodb\_data\_file\_path](#_bookmark45)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-data-file-path=file\_name |
| System Variable | [innodb\_data\_file\_path](#_bookmark45) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | String |
| Default Value | ibdata1:12M:autoextend |

Defines the name, size, and attributes of InnoDB system tablespace data files. If you do not specify a value for [innodb\_data\_file\_path](#_bookmark45), 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](#_bookmark142)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-data-home-dir=dir\_name |
| System Variable | [innodb\_data\_home\_dir](#_bookmark142) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |

The common part of the directory path for InnoDB system tablespace data files. The default value is the MySQL data directory. The setting is concatenated with the [innodb\_data\_file\_path](#_bookmark45) setting, unless that setting is defined with an absolute path.

A trailing slash is required when specifying a value for [innodb\_data\_home\_dir](#_bookmark142). 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 15.8.1, “InnoDB Startup Configuration” .

• [innodb\_ddl\_buffer\_size](#_bookmark74)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ddl-buffer-size=# |
| Introduced | 8.0.27 |
| System Variable | [innodb\_ddl\_buffer\_size](#_bookmark74) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1048576 |
| Minimum Value | 65536 |
| Maximum Value | 4294967295 |
| Unit | bytes |

Defines the maximum buffer size for DDL operations. The default setting is 1048576 bytes (approximately 1 MB). Applies to online DDL operations that create or rebuild secondary indexes. See [Section 15.12.4, “Online DDL Memory Management”](#_bookmark73) . The maximum buffer size per DDL thread is the maximum buffer size divided by the number of DDL threads ([innodb\_ddl\_buffer\_size](#_bookmark74)/[innodb\_ddl\_threads](#_bookmark75)).

• [innodb\_ddl\_log\_crash\_reset\_debug](#_bookmark141)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ddl-log-crash-reset-  debug[={OFF |ON}] |

|  |  |
| --- | --- |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

• [innodb\_ddl\_threads](#_bookmark75)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ddl-threads=# |
| Introduced | 8.0.27 |
| System Variable | [innodb\_ddl\_threads](#_bookmark75) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 4 |
| Minimum Value | 1 |
| Maximum Value | 64 |

Defines the maximum number of parallel threads for the sort and build phases of index creation. Applies to online DDL operations that create or rebuild secondary indexes. For related information, see [Section 15.12.5, “Configuring Parallel Threads for Online DDL Operations”](#_bookmark76) , and [Section 15.12.4,](#_bookmark73) [“Online DDL Memory Management”](#_bookmark73) .

• [innodb\_deadlock\_detect](#_bookmark143)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-deadlock-detect[={OFF |ON}] |
| System Variable | [innodb\_deadlock\_detect](#_bookmark143) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

This option is used to disable deadlock detection. On high concurrency systems, deadlock detection can cause a slowdown when numerous threads wait for the same lock. At times, it may be more efficient to disable deadlock detection and rely on the [innodb\_lock\_wait\_timeout](#_bookmark144) setting for transaction rollback when a deadlock occurs.

For related information, see Section 15.7.5.2, “Deadlock Detection” .

• [innodb\_dedicated\_server](#_bookmark13)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-dedicated-server[={OFF |ON}] |
| System Variable | [innodb\_dedicated\_server](#_bookmark13) |
| Scope | Global |



|  |  |
| --- | --- |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

When [innodb\_dedicated\_server](#_bookmark13) is enabled, InnoDB automatically configures the following variables:

• [innodb\_buffer\_pool\_size](#_bookmark14)

• [innodb\_redo\_log\_capacity](#_bookmark15) or, prior to MySQL 8.0.30, [innodb\_log\_file\_size](#_bookmark10) and [innodb\_log\_files\_in\_group](#_bookmark11).

**Note**

innodb\_log\_file\_size and innodb\_log\_files\_in\_group are deprecated in MySQL 8.0.30. These variables are superseded by innodb\_redo\_log\_capacity. For more information, see Section 15.6.5, “Redo Log” .

• [innodb\_flush\_method](#_bookmark12)

Only consider enabling [innodb\_dedicated\_server](#_bookmark13) if the MySQL instance resides on a dedicated server where it can use all available system resources. Enabling [innodb\_dedicated\_server](#_bookmark13) is not recommended if the MySQL instance shares system resources with other applications.

For more information, see [Section 15.8.12, “Enabling Automatic Configuration for a Dedicated](#_bookmark9) [MySQL Server”](#_bookmark9) .

• [innodb\_default\_row\_format](#_bookmark42)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-default-row-format=value |
| System Variable | [innodb\_default\_row\_format](#_bookmark42) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark42) 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](#_bookmark42). For more information, see [Defining the Row Format of a Table](#_bookmark39).

Internal InnoDB temporary tables created by the server to process queries use the DYNAMIC row format, regardless of the [innodb\_default\_row\_format](#_bookmark42) setting.

• [innodb\_directories](#_bookmark145)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-directories=dir\_name |
| System Variable | [innodb\_directories](#_bookmark145) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |
| Default Value | NULL |

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

Tablespace discovery during crash recovery relies on the [innodb\_directories](#_bookmark145) setting to identify tablespaces referenced in the redo logs. For more information, see Tablespace Discovery During Crash Recovery.

The default value is NULL, but directories defined by [innodb\_data\_home\_dir](#_bookmark142), [innodb\_undo\_directory](#_bookmark146), and datadir are always appended to the [innodb\_directories](#_bookmark145) argument value when InnoDB builds a list of directories to scan at startup. These directories are appended regardless of whether an [innodb\_directories](#_bookmark145) setting is specified explicitly.

[innodb\_directories](#_bookmark145) may be specified as an option in a startup command or in a MySQL option file. Quotes surround the argument value because otherwise some command interpreters interpret semicolon (;) as a special character. (For example, Unix shells treat it as a command terminator.)

Startup command:

mysqld --innodb-directories="*directory\_path\_1*;*directory\_path\_2*"

MySQL option file:

[mysqld]

innodb\_directories="*directory\_path\_1*;*directory\_path\_2*"

Wildcard expressions cannot be used to specify directories.

The [innodb\_directories](#_bookmark145) scan also traverses the subdirectories of specified directories. Duplicate directories and subdirectories are discarded from the list of directories to be scanned.

For more information, see Section 15.6.3.6, “Moving Tablespace Files While the Server is Offline” .

• [innodb\_disable\_sort\_file\_cache](#_bookmark147)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-disable-sort-file-  cache[={OFF |ON}] |
| System Variable | [innodb\_disable\_sort\_file\_cache](#_bookmark147) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

• [innodb\_doublewrite](#_bookmark44)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-doublewrite=value (≥ 8.0.30)  --innodb-doublewrite[={OFF |ON}] (≤ 8.0.29) |
| System Variable | [innodb\_doublewrite](#_bookmark44) |
| Scope | Global |
| Dynamic (≥ 8.0.30) | Yes |
| Dynamic (≤ 8.0.29) | No |
| SET\_VAR Hint Applies | No |
| Type (≥ 8.0.30) | Enumeration |
| Type (≤ 8.0.29) | Boolean |
| Default Value | ON |
| Valid Values | ON  OFF  DETECT\_AND\_RECOVER  DETECT\_ONLY |

The [innodb\_doublewrite](#_bookmark44) variable controls doublewrite buffering. Doublewrite buffering is enabled by default in most cases.

Prior to MySQL 8.0.30, you can set [innodb\_doublewrite](#_bookmark44) to ON or OFF when starting the server to enable or disable doublewrite buffering, respectively. From MySQL 8.0.30, [innodb\_doublewrite](#_bookmark44) also supports DETECT\_AND\_RECOVER and DETECT\_ONLY settings.

The DETECT\_AND\_RECOVER setting is the same as the ON setting. With this setting, the doublewrite buffer is fully enabled, with database page content written to the doublewrite buffer where it is accessed during recovery to fix incomplete page writes.

With the DETECT\_ONLY setting, only metadata is written to the doublewrite buffer. Database page content is not written to the doublewrite buffer, and recovery does not use the doublewrite buffer to fix incomplete page writes. This lightweight setting is intended for detecting incomplete page writes only.

MySQL 8.0.30 onwards supports dynamic changes to the [innodb\_doublewrite](#_bookmark44) setting that enables the doublewrite buffer, between ON, DETECT\_AND\_RECOVER, and DETECT\_ONLY. MySQL

does not support dynamic changes between a setting that enables the doublewrite buffer and OFF or vice versa.

If the doublewrite buffer is located on a Fusion-io device that supports atomic writes, the doublewrite buffer is automatically disabled and data file writes are performed using Fusion-io atomic writes instead. However, be aware that the [innodb\_doublewrite](#_bookmark44) setting is global. When the doublewrite buffer is disabled, it is disabled for all data files including those that do not reside on Fusion-io hardware. This feature is only supported on Fusion-io hardware and is only enabled for Fusion- io NVMFS on Linux. To take full advantage of this feature, an [innodb\_flush\_method](#_bookmark12) setting of O\_DIRECT is recommended.

For related information, see Section 15.6.4, “Doublewrite Buffer” .

•

•

•

[innodb\_doublewrite\_batch\_size](#_bookmark148)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-doublewrite-batch-size=# |
| Introduced | 8.0.20 |
| System Variable | [innodb\_doublewrite\_batch\_size](#_bookmark148) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 256 |

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

For more information, see Section 15.6.4, “Doublewrite Buffer” .

[innodb\_doublewrite\_dir](#_bookmark149)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-doublewrite-dir=dir\_name |
| Introduced | 8.0.20 |
| System Variable | [innodb\_doublewrite\_dir](#_bookmark149) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |

Defines the directory for doublewrite files. If no directory is specified, doublewrite files are created in the [innodb\_data\_home\_dir](#_bookmark142) directory, which defaults to the data directory if unspecified.

For more information, see Section 15.6.4, “Doublewrite Buffer” .

[innodb\_doublewrite\_files](#_bookmark150)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-doublewrite-files=# |
| Introduced | 8.0.20 |
| System Variable | [innodb\_doublewrite\_files](#_bookmark150) |
| Scope | Global |
| Dynamic | No |

|  |  |
| --- | --- |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | innodb\_buffer\_pool\_instances \* 2 |
| Minimum Value | 2 |
| Maximum Value | 256 |

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

At a minimum, there are two doublewrite files. The maximum number of doublewrite files is two times the number of buffer pool instances. (The number of buffer pool instances is controlled by the [innodb\_buffer\_pool\_instances](#_bookmark122) variable.)

For more information, see Section 15.6.4, “Doublewrite Buffer” .

• [innodb\_doublewrite\_pages](#_bookmark151)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-doublewrite-pages=# |
| Introduced | 8.0.20 |
| System Variable | [innodb\_doublewrite\_pages](#_bookmark151) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | innodb\_write\_io\_threads value |
| Minimum Value | innodb\_write\_io\_threads value |
| Maximum Value | 512 |

Defines the maximum number of doublewrite pages per thread for a batch write. If no value is specified, [innodb\_doublewrite\_pages](#_bookmark151) is set to the [innodb\_write\_io\_threads](#_bookmark152) value.

For more information, see Section 15.6.4, “Doublewrite Buffer” .

• [innodb\_extend\_and\_initialize](#_bookmark153)

|  |  |
| --- | --- |
| Command-Line Format | --innodb=extend-and-  initialize[={OFF |ON}] |
| Introduced | 8.0.22 |
| System Variable | [innodb\_extend\_and\_initialize](#_bookmark153) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

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

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

• [innodb\_fast\_shutdown](#_bookmark154)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-fast-shutdown=# |
| System Variable | [innodb\_fast\_shutdown](#_bookmark154) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark154) in emergency or troubleshooting situations, to get the absolute fastest shutdown if data is at risk of corruption.

• [innodb\_fil\_make\_page\_dirty\_debug](#_bookmark155)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-fil-make-page-dirty-debug=# |
| System Variable | [innodb\_fil\_make\_page\_dirty\_debug](#_bookmark155) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 2\*\*32-1 |

By default, setting [innodb\_fil\_make\_page\_dirty\_debug](#_bookmark155) to the ID of a tablespace immediately dirties the first page of the tablespace. If [innodb\_saved\_page\_number\_debug](#_bookmark156) is set to a non- default value, setting [innodb\_fil\_make\_page\_dirty\_debug](#_bookmark155) dirties the specified page. The [innodb\_fil\_make\_page\_dirty\_debug](#_bookmark155) option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

• [innodb\_file\_per\_table](#_bookmark17)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-file-per-table[={OFF |ON}] |
| System Variable | [innodb\_file\_per\_table](#_bookmark17) |

|  |  |
| --- | --- |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

When [innodb\_file\_per\_table](#_bookmark17) 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 15.6.3.2, “File-Per-Table Tablespaces” . For information about the InnoDB system tablespace, see Section 15.6.3.1, “The System Tablespace” .

The [innodb\_file\_per\_table](#_bookmark17) 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 5.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

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

The [innodb\_file\_per-table](#_bookmark17) setting does not affect the creation of temporary tables. As of MySQL 8.0.14, temporary tables are created in session temporary tablespaces, and in the global temporary tablespace before that. See Section 15.6.3.5, “Temporary Tablespaces” .

• [innodb\_fill\_factor](#_bookmark157)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-fill-factor=# |
| System Variable | [innodb\_fill\_factor](#_bookmark157) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 100 |
| Minimum Value | 10 |
| Maximum Value | 100 |

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

[innodb\_fill\_factor](#_bookmark157) 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](#_bookmark157) 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](#_bookmark157) setting is interpreted as a hint rather than a hard limit.

An [innodb\_fill\_factor](#_bookmark157) setting of 100 leaves 1/16 of the space in clustered index pages free for future index growth.

[innodb\_fill\_factor](#_bookmark157) 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 15.6.2.3, “Sorted Index Builds” .

• [innodb\_flush\_log\_at\_timeout](#_bookmark158)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flush-log-at-timeout=# |
| System Variable | [innodb\_flush\_log\_at\_timeout](#_bookmark158) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1 |
| Minimum Value | 1 |
| Maximum Value | 2700 |
| Unit | seconds |

Write and flush the logs every *N*seconds. [innodb\_flush\_log\_at\_timeout](#_bookmark158) 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](#_bookmark158) is once per second.

• [innodb\_flush\_log\_at\_trx\_commit](#_bookmark159)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flush-log-at-trx-commit=# |
| System Variable | [innodb\_flush\_log\_at\_trx\_commit](#_bookmark159) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark159) 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](#_bookmark158), 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](#_bookmark159) setting.

• InnoDB crash recovery works regardless of the [innodb\_flush\_log\_at\_trx\_commit](#_bookmark159) 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](#_bookmark159).

For information on the combination of settings on a replica that is most resilient to unexpected halts, see Section 17.4.2, “Handling an Unexpected Halt of a Replica” .

**Caution**

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

[innodb\_flush\_method](#_bookmark12)

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flush-method=value |
| System Variable | [innodb\_flush\_method](#_bookmark12) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | String |
| Default Value (Unix) | fsync |
| Default Value (Windows) | unbuffered |
| Valid Values (Unix) | fsync  O\_DSYNC  littlesync  nosync  O\_DIRECT  O\_DIRECT\_NO\_FSYNC |
| Valid Values (Windows) | unbuffered |



|  |  |
| --- | --- |
|  | normal |

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

On Unix-like systems, the default value is fsync. On Windows, the default value is unbuffered.

**Note**

In MySQL 8.0, [innodb\_flush\_method](#_bookmark12) options can be specified numerically.

The [innodb\_flush\_method](#_bookmark12) options for Unix-like systems include:

• fsync or 0: InnoDB uses the fsync() system call to flush both the data and log files. fsync is the default setting.

• O\_DSYNC or 1: InnoDB uses O\_SYNC to open and flush the log files, and fsync() to flush the data files. InnoDB does not use O\_DSYNC directly because there have been problems with it on many varieties of Unix.

• littlesync or 2: This option is used for internal performance testing and is currently unsupported. Use at your own risk.

• nosync or 3: This option is used for internal performance testing and is currently unsupported. Use at your own risk.

• O\_DIRECT or 4: InnoDB uses O\_DIRECT (or directio() on Solaris) to open the data files, and uses fsync() to flush both the data and log files. This option is available on some GNU/Linux versions, FreeBSD, and Solaris.

• O\_DIRECT\_NO\_FSYNC: InnoDB uses O\_DIRECT during flushing I/O, but skips the fsync() system call after each write operation.

Prior to MySQL 8.0.14, 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 8.0.14, 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.

On platforms that support fdatasync() system calls, the [innodb\_use\_fdatasync](#_bookmark160) variable, introduced in MySQL 8.0.26, permits [innodb\_flush\_method](#_bookmark12) options that use fsync() to use fdatasync() instead. An fdatasync() system call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

The [innodb\_flush\_method](#_bookmark12) options for Windows systems include:

• unbuffered or 0: InnoDB uses simulated asynchronous I/O and non-buffered I/O.

• normal or 1: InnoDB uses simulated asynchronous I/O and buffered I/O.

How each setting affects performance depends on hardware configuration and workload. Benchmark your particular configuration to decide which setting to use, or whether to keep the default setting. Examine the Innodb\_data\_fsyncs status variable to see the overall number of fsync() calls (or fdatasync() calls if [innodb\_use\_fdatasync](#_bookmark160) is enabled) for each setting. The mix of read and write operations in your workload can affect how a setting performs. For example, on a system with a hardware RAID controller and battery-backed write cache, O\_DIRECT can help to avoid double buffering between the InnoDB buffer pool and the operating system file system cache. On some systems where InnoDB data and log files are located on a SAN, the default value or O\_DSYNC might be faster for a read-heavy workload with mostly SELECT statements. Always test this parameter with hardware and workload that reflect your production environment. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O” .

If [innodb\_dedicated\_server](#_bookmark13) is enabled, the [innodb\_flush\_method](#_bookmark12) value is automatically configured if it is not explicitly defined. For more information, see [Section 15.8.12, “Enabling](#_bookmark9) [Automatic Configuration for a Dedicated MySQL Server”](#_bookmark9) .

• [innodb\_flush\_neighbors](#_bookmark161)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flush-neighbors=# |
| System Variable | [innodb\_flush\_neighbors](#_bookmark161) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Enumeration |
| Default Value | 0 |
| 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](#_bookmark161). Dirty pages in the same extent are not flushed.

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

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

When the table data is stored on a traditional HDD storage device, flushing such neighbor pages in one operation reduces I/O overhead (primarily for disk seek operations) compared to flushing individual pages at different times. For table data stored on SSD, seek time is not a significant factor and you can set this option to 0 to spread out write operations. For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing” .

• [innodb\_flush\_sync](#_bookmark162)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flush-sync[={OFF |ON}] |
| System Variable | [innodb\_flush\_sync](#_bookmark162) |
| Scope | Global |
| Dynamic | Yes |

|  |  |
| --- | --- |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

The [innodb\_flush\_sync](#_bookmark162) variable, which is enabled by default, causes the [innodb\_io\_capacity](#_bookmark163) 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](#_bookmark163) setting, disable [innodb\_flush\_sync](#_bookmark162).

For information about configuring the [innodb\_flush\_sync](#_bookmark162) variable, see Section 15.8.7, “Configuring InnoDB I/O Capacity” .

• [innodb\_flushing\_avg\_loops](#_bookmark164)

•

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-flushing-avg-loops=# |
| System Variable | [innodb\_flushing\_avg\_loops](#_bookmark164) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 30 |
| Minimum Value | 1 |
| Maximum Value | 1000 |

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

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing” .

[innodb\_force\_load\_corrupted](#_bookmark165)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-force-load-corrupted[={OFF |  ON}] |
| System Variable | [innodb\_force\_load\_corrupted](#_bookmark165) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

[innodb\_force\_recovery](#_bookmark166)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-force-recovery=# |
| System Variable | [innodb\_force\_recovery](#_bookmark166) |
| Scope | Global |



|  |  |
| --- | --- |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 6 |

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. For the meanings of these values and important information about [innodb\_force\_recovery](#_bookmark166), see Section 15.21.3, “Forcing InnoDB Recovery” .

**Warning**

Only set this variable to a value greater than 0 in an emergency situation so that you can start InnoDB and dump your tables. As a safety measure, InnoDB prevents INSERT, UPDATE, or DELETE

operations when [innodb\_force\_recovery](#_bookmark166) is greater than 0. An [innodb\_force\_recovery](#_bookmark166) setting of 4 or greater places InnoDB into read- only mode.

These restrictions may cause replication administration commands to fail with an error, as replication stores the replica status logs in InnoDB tables.

• [innodb\_fsync\_threshold](#_bookmark167)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-fsync-threshold=# |
| Introduced | 8.0.13 |
| System Variable | [innodb\_fsync\_threshold](#_bookmark167) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 2\*\*64-1 |

By default, when InnoDB creates a new data file, such as a new log file or tablespace file, the

file is fully written to the operating system cache before it is flushed to disk, which can cause a large amount of disk write activity to occur at once. To force smaller, periodic flushes of data from the operating system cache, you can use the [innodb\_fsync\_threshold](#_bookmark167) variable to define a threshold value, in bytes. When the byte threshold is reached, the contents of the operating system cache are flushed to disk. The default value of 0 forces the default behavior, which is to flush data to disk only after a file is fully written to the cache.

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

• [innodb\_ft\_aux\_table](#_bookmark168)



System Variable innodb\_ft\_aux\_table

|  |  |
| --- | --- |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | String |

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

SET GLOBAL innodb\_ft\_aux\_table = 'test/t1';

After you set this variable to a name in the format *db\_name*/*table\_name*, the INFORMATION\_SCHEMA tables INNODB\_FT\_INDEX\_TABLE, INNODB\_FT\_INDEX\_CACHE, INNODB\_FT\_CONFIG, INNODB\_FT\_DELETED, and INNODB\_FT\_BEING\_DELETED show information about the search index for the specified table.

For more information, see [Section 15.15.4, “InnoDB INFORMATION\_SCHEMA FULLTEXT Index](#_bookmark169) [Tables”](#_bookmark169) .

• [innodb\_ft\_cache\_size](#_bookmark170)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-cache-size=# |
| System Variable | [innodb\_ft\_cache\_size](#_bookmark170) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 8000000 |
| Minimum Value | 1600000 |
| Maximum Value | 80000000 |
| Unit | bytes |

The memory allocated, in bytes, for the InnoDB FULLTEXT search index cache, which holds a parsed document in memory while creating an InnoDB FULLTEXT index. Index inserts and updates are only committed to disk when the [innodb\_ft\_cache\_size](#_bookmark170) size limit is reached. [innodb\_ft\_cache\_size](#_bookmark170) defines the cache size on a per table basis. To set a global limit for all tables, see [innodb\_ft\_total\_cache\_size](#_bookmark171).

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

• [innodb\_ft\_enable\_diag\_print](#_bookmark172)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-enable-diag-print[={OFF |  ON}] |
| System Variable | [innodb\_ft\_enable\_diag\_print](#_bookmark172) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |

|  |  |
| --- | --- |
| Default Value | OFF |

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

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

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

SYNC words: 100

• FTS optimize progress. For example:

FTS start optimize test

FTS\_OPTIMIZE: optimize "mysql"

FTS\_OPTIMIZE: processed "mysql"

• FTS index build progress. For example:

Number of doc processed: 1000

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

FTS Search Processing time: 1 secs: 100 millisec: row(s) 10000

Full Search Memory: 245666 (bytes), Row: 10000

• [innodb\_ft\_enable\_stopword](#_bookmark173)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-enable-stopword[={OFF |  ON}] |
| System Variable | [innodb\_ft\_enable\_stopword](#_bookmark173) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Specifies that a set of stopwords is associated with an InnoDB FULLTEXT index at the time the index is created. If the [innodb\_ft\_user\_stopword\_table](#_bookmark174) option is set, the stopwords are taken from that table. Else, if the [innodb\_ft\_server\_stopword\_table](#_bookmark175) 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](#_bookmark176)

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-max-token-size=# |
| System Variable | [innodb\_ft\_max\_token\_size](#_bookmark176) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 84 |
| Minimum Value | 10 |
| Maximum Value | 84 |

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

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search” .

• [innodb\_ft\_min\_token\_size](#_bookmark177)

•

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-min-token-size=# |
| System Variable | [innodb\_ft\_min\_token\_size](#_bookmark177) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 3 |
| Minimum Value | 0 |
| Maximum Value | 16 |

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

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search” .

[innodb\_ft\_num\_word\_optimize](#_bookmark178)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-num-word-optimize=# |
| System Variable | [innodb\_ft\_num\_word\_optimize](#_bookmark178) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 2000 |
| Minimum Value | 1000 |
| Maximum Value | 10000 |

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

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search” .

[innodb\_ft\_result\_cache\_limit](#_bookmark179)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-result-cache-limit=# |
| System Variable | [innodb\_ft\_result\_cache\_limit](#_bookmark179) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |

|  |  |
| --- | --- |
| Type | Integer |
| Default Value | 2000000000 |
| Minimum Value | 1000000 |
| Maximum Value | 2\*\*32-1 |
| Unit | bytes |

The InnoDB full-text search query result cache limit (defined in bytes) per full-text search query or per thread. Intermediate and final InnoDB full-text search query results are handled in memory. Use [innodb\_ft\_result\_cache\_limit](#_bookmark179) 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](#_bookmark179) for all platform types and bit sizes is

2\*\*32-1.

• [innodb\_ft\_server\_stopword\_table](#_bookmark175)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-server-stopword-  table=db\_name/table\_name |
| System Variable | [innodb\_ft\_server\_stopword\_table](#_bookmark175) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | String |
| Default Value | NULL |

This option is used to specify your own InnoDB FULLTEXT index stopword list for all InnoDB tables. To configure your own stopword list for a specific InnoDB table, use [innodb\_ft\_user\_stopword\_table](#_bookmark174).

Set [innodb\_ft\_server\_stopword\_table](#_bookmark175) 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](#_bookmark175). [innodb\_ft\_enable\_stopword](#_bookmark173) must be enabled and [innodb\_ft\_server\_stopword\_table](#_bookmark175) 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](#_bookmark180)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-sort-pll-degree=# |
| System Variable | [innodb\_ft\_sort\_pll\_degree](#_bookmark180) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.6.2.4, “InnoDB Full-Text Indexes” , and

[innodb\_sort\_buffer\_size](#_bookmark70).

• [innodb\_ft\_total\_cache\_size](#_bookmark171)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-total-cache-size=# |
| System Variable | [innodb\_ft\_total\_cache\_size](#_bookmark171) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 640000000 |
| Minimum Value | 32000000 |
| Maximum Value | 1600000000 |
| Unit | bytes |

The total memory allocated, in bytes, for the InnoDB full-text search index cache for all tables. Creating numerous tables, each with a FULLTEXT search index, could consume a significant portion of available memory. [innodb\_ft\_total\_cache\_size](#_bookmark171) 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](#_bookmark174)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-ft-user-stopword-  table=db\_name/table\_name |
| System Variable | [innodb\_ft\_user\_stopword\_table](#_bookmark174) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | String |

|  |  |
| --- | --- |
| Default Value | NULL |

This option is used to specify your own InnoDB FULLTEXT index stopword list on a specific table. To configure your own stopword list for all InnoDB tables, use [innodb\_ft\_server\_stopword\_table](#_bookmark175).

Set [innodb\_ft\_user\_stopword\_table](#_bookmark174) 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](#_bookmark174). [innodb\_ft\_enable\_stopword](#_bookmark173) must be enabled and [innodb\_ft\_user\_stopword\_table](#_bookmark174) 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\_idle\_flush\_pct](#_bookmark181)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-idle-flush-pct=# |
| Introduced | 8.0.18 |
| System Variable | [innodb\_idle\_flush\_pct](#_bookmark181) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 100 |
| Minimum Value | 0 |
| Maximum Value | 100 |

Limits page flushing when InnoDB is idle. The [innodb\_idle\_flush\_pct](#_bookmark181) value is a percentage of the [innodb\_io\_capacity](#_bookmark163) setting, which defines the number of I/O operations per second available to InnoDB. For more information, see Limiting Buffer Flushing During Idle Periods.

• [innodb\_io\_capacity](#_bookmark163)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-io-capacity=# |
| System Variable | [innodb\_io\_capacity](#_bookmark163) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark163) 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](#_bookmark163) variable, see Section 15.8.7, “Configuring InnoDB I/O Capacity” .

• [innodb\_io\_capacity\_max](#_bookmark182)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-io-capacity-max=# |
| System Variable | [innodb\_io\_capacity\_max](#_bookmark182) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark163) variable. The [innodb\_io\_capacity\_max](#_bookmark182) variable defines a maximum number of IOPS performed by InnoDB background tasks in such situations.

For information about configuring the [innodb\_io\_capacity\_max](#_bookmark182) variable, see Section 15.8.7, “Configuring InnoDB I/O Capacity” .

• [innodb\_limit\_optimistic\_insert\_debug](#_bookmark183)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-limit-optimistic-insert-  debug=# |
| System Variable | [innodb\_limit\_optimistic\_insert\_debug](#_bookmark183) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 2\*\*32-1 |

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

• [innodb\_lock\_wait\_timeout](#_bookmark144)

|  |  |
| --- | --- |
| System Variable | [innodb\_lock\_wait\_timeout](#_bookmark144) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50 |
| Minimum Value | 1 |
| Maximum Value | 1073741824 |
| Unit | seconds |

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

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

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the [--innodb-rollback-on-timeout](#_bookmark184) option. See also Section 15.21.5, “InnoDB Error Handling” .

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

[innodb\_lock\_wait\_timeout](#_bookmark144) applies to InnoDB row locks. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks when [innodb\_deadlock\_detect](#_bookmark143) is enabled (the default) because InnoDB detects deadlocks immediately and rolls back one of the deadlocked transactions. When [innodb\_deadlock\_detect](#_bookmark143) is disabled, InnoDB relies on [innodb\_lock\_wait\_timeout](#_bookmark144) for transaction rollback when a deadlock occurs. See

Section 15.7.5.2, “Deadlock Detection” .

[innodb\_lock\_wait\_timeout](#_bookmark144) 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. 1.9. 1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the SESSION setting for [innodb\_lock\_wait\_timeout](#_bookmark144), which affects only that client.

• [innodb\_log\_buffer\_size](#_bookmark185)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-buffer-size=# |
| System Variable | [innodb\_log\_buffer\_size](#_bookmark185) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | 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 is 16MB. 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\_fuzzy\_now](#_bookmark186)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-checkpoint-fuzzy-  now[={OFF |ON}] |
| Introduced | 8.0.13 |
| System Variable | [innodb\_log\_checkpoint\_fuzzy\_now](#_bookmark186) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

• [innodb\_log\_checkpoint\_now](#_bookmark187)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-checkpoint-now[={OFF |  ON}] |
| System Variable | [innodb\_log\_checkpoint\_now](#_bookmark187) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enable this debug option to force InnoDB to write a checkpoint. This option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

• [innodb\_log\_checksums](#_bookmark188)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-checksums[={OFF |ON}] |
| System Variable | [innodb\_log\_checksums](#_bookmark188) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |

|  |  |
| --- | --- |
| Default Value | ON |

Enables or disables checksums for redo log pages.

[innodb\_log\_checksums=ON](#_bookmark188) enables the CRC-32C checksum algorithm for redo log pages. When [innodb\_log\_checksums](#_bookmark188) 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](#_bookmark31)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-compressed-pages[={OFF |  ON}] |
| System Variable | [innodb\_log\_compressed\_pages](#_bookmark31) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

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

[innodb\_log\_compressed\_pages](#_bookmark31) 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](#_bookmark31) to reduce redo log generation for workloads that modify compressed data.

To measure the effect of enabling or disabling [innodb\_log\_compressed\_pages](#_bookmark31), 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 15.9.1.6, “Compression for OLTP Workloads”](#_bookmark28) .

• [innodb\_log\_file\_size](#_bookmark10)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-file-size=# |
| Deprecated | 8.0.30 |
| System Variable | [innodb\_log\_file\_size](#_bookmark10) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50331648 |
| Minimum Value | 4194304 |
| Maximum Value | 512GB / innodb\_log\_files\_in\_group |



|  |  |
| --- | --- |
| Unit | bytes |

**Note**

[innodb\_log\_file\_size](#_bookmark10) and [innodb\_log\_files\_in\_group](#_bookmark11) are deprecated in MySQL 8.0.30. These variables are superseded by [innodb\_redo\_log\_capacity](#_bookmark15). For more information, see Section 15.6.5, “Redo Log” .

The size in bytes of each log file in a log group. The combined size of log files ([innodb\_log\_file\_size](#_bookmark10) \* [innodb\_log\_files\_in\_group](#_bookmark11)) 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](#_bookmark10) is 4MB.

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

If [innodb\_dedicated\_server](#_bookmark13) is enabled, the [innodb\_log\_file\_size](#_bookmark10) value is automatically configured if it is not explicitly defined. For more information, see [Section 15.8.12, “Enabling](#_bookmark9) [Automatic Configuration for a Dedicated MySQL Server”](#_bookmark9) .

• [innodb\_log\_files\_in\_group](#_bookmark11)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-files-in-group=# |
| Deprecated | 8.0.30 |
| System Variable | [innodb\_log\_files\_in\_group](#_bookmark11) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 2 |
| Minimum Value | 2 |
| Maximum Value | 100 |

**Note**

[innodb\_log\_file\_size](#_bookmark10) and [innodb\_log\_files\_in\_group](#_bookmark11) are deprecated in MySQL 8.0.30. These variables are superseded by [innodb\_redo\_log\_capacity](#_bookmark15). For more information, see Section 15.6.5, “Redo Log” .

The number of log files in the log group. InnoDB writes to the files in a circular fashion. The default (and recommended) value is 2. The location of the files is specified by

[innodb\_log\_group\_home\_dir](#_bookmark189). The combined size of log files ([innodb\_log\_file\_size](#_bookmark10) \*

[innodb\_log\_files\_in\_group](#_bookmark11)) can be up to 512GB.

For related information, see Redo Log Configuration.

If [innodb\_dedicated\_server](#_bookmark13) is enabled, [innodb\_log\_files\_in\_group](#_bookmark11) is automatically configured if it is not explicitly defined. For more information, see [Section 15.8.12, “Enabling](#_bookmark9) [Automatic Configuration for a Dedicated MySQL Server”](#_bookmark9) .

• [innodb\_log\_group\_home\_dir](#_bookmark189)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-group-home-dir=dir\_name |
| System Variable | [innodb\_log\_group\_home\_dir](#_bookmark189) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |

The directory path to the InnoDB redo log files.

For related information, see Redo Log Configuration.

• [innodb\_log\_spin\_cpu\_abs\_lwm](#_bookmark190)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-spin-cpu-abs-lwm=# |
| System Variable | [innodb\_log\_spin\_cpu\_abs\_lwm](#_bookmark190) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 80 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |

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

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging” .

• [innodb\_log\_spin\_cpu\_pct\_hwm](#_bookmark191)

|  |  |  |
| --- | --- | --- |
|  | Command-Line Format | --innodb-log-spin-cpu-pct-hwm=# |
| System Variable | [innodb\_log\_spin\_cpu\_pct\_hwm](#_bookmark191) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50 |
| 3294 | Minimum Value | 0 |

|  |  |
| --- | --- |
| Maximum Value | 100 |

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

The [innodb\_log\_spin\_cpu\_pct\_hwm](#_bookmark191) variable respects processor affinity. For example, if a server has 48 cores but the mysqld process is pinned to only four CPU cores, the other 44 CPU cores are ignored.

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging” .

• [innodb\_log\_wait\_for\_flush\_spin\_hwm](#_bookmark192)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-wait-for-flush-spin-  hwm=# |
| System Variable | [innodb\_log\_wait\_for\_flush\_spin\_hwm](#_bookmark192) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 400 |
| Minimum Value | 0 |
| Maximum Value (64-bit platforms) | 2\*\*64-1 |
| Maximum Value (32-bit platforms) | 2\*\*32-1 |
| Unit | microseconds |

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

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging” .

• [innodb\_log\_write\_ahead\_size](#_bookmark193)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-write-ahead-size=# |
| System Variable | [innodb\_log\_write\_ahead\_size](#_bookmark193) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 8192 |
| Minimum Value | 512 (log file block size) |
| Maximum Value | Equal to innodb\_page\_size |
| Unit | bytes |

Defines the write-ahead block size for the redo log, in bytes. To avoid “read-on-write” , set [innodb\_log\_write\_ahead\_size](#_bookmark193) 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](#_bookmark193) are multiples of the InnoDB log file block

size (2n). 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](#_bookmark7)

value. If you specify a value for [innodb\_log\_write\_ahead\_size](#_bookmark193) that is larger than the

[innodb\_page\_size](#_bookmark7) value, the [innodb\_log\_write\_ahead\_size](#_bookmark193) setting is truncated to the

[innodb\_page\_size](#_bookmark7) value.

Setting the [innodb\_log\_write\_ahead\_size](#_bookmark193) 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\_log\_writer\_threads](#_bookmark194)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-log-writer-threads[={OFF |  ON}] |
| Introduced | 8.0.22 |
| System Variable | [innodb\_log\_writer\_threads](#_bookmark194) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

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

For more information, see Section 8.5.4, “Optimizing InnoDB Redo Logging” .

• [innodb\_lru\_scan\_depth](#_bookmark195)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-lru-scan-depth=# |
| System Variable | [innodb\_lru\_scan\_depth](#_bookmark195) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark195), 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](#_bookmark195) when changing the number of buffer pool instances, since [innodb\_lru\_scan\_depth](#_bookmark195) \* [innodb\_buffer\_pool\_instances](#_bookmark122) defines the amount of work performed by the page cleaner thread each second.

For related information, see Section 15.8.3.5, “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](#_bookmark196)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-max-dirty-pages-pct=# |
| System Variable | [innodb\_max\_dirty\_pages\_pct](#_bookmark196) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Numeric |
| Default Value | 90 |
| 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 [innodb\_max\_dirty\_pages\_pct](#_bookmark196) 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 15.8.3.5, “Configuring Buffer Pool Flushing” .

For related information, see Section 15.8.3.5, “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](#_bookmark197)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-max-dirty-pages-pct-lwm=# |
| System Variable | [innodb\_max\_dirty\_pages\_pct\_lwm](#_bookmark197) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Numeric |
| Default Value | 10 |
| Minimum Value | 0 |
| Maximum Value | 99.99 |

Defines a low water mark representing the percentage of dirty pages at which preflushing is

•

•

•

configured value should always be lower than the [innodb\_max\_dirty\_pages\_pct](#_bookmark196) value. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing” .

[innodb\_max\_purge\_lag](#_bookmark198)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-max-purge-lag=# |
| System Variable | [innodb\_max\_purge\_lag](#_bookmark198) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |

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

For more information, see Section 15.8.9, “Purge Configuration” .

[innodb\_max\_purge\_lag\_delay](#_bookmark199)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-max-purge-lag-delay=# |
| System Variable | [innodb\_max\_purge\_lag\_delay](#_bookmark199) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 10000000 |
| Unit | microseconds |

Specifies the maximum delay in microseconds for the delay imposed when the [innodb\_max\_purge\_lag](#_bookmark198) threshold is exceeded. The specified [innodb\_max\_purge\_lag\_delay](#_bookmark199) value is an upper limit on the delay period calculated by the [innodb\_max\_purge\_lag](#_bookmark198) formula.

For more information, see Section 15.8.9, “Purge Configuration” .

[innodb\_max\_undo\_log\_size](#_bookmark200)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-max-undo-log-size=# |
| System Variable | [innodb\_max\_undo\_log\_size](#_bookmark200) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1073741824 |
| Minimum Value | 10485760 |

|  |  |
| --- | --- |
| Maximum Value | 2\*\*64-1 |
| Unit | bytes |

Defines a threshold size for undo tablespaces. If an undo tablespace exceeds the threshold, it can be marked for truncation when [innodb\_undo\_log\_truncate](#_bookmark201) is enabled. The default value is 1073741824 bytes (1024 MiB).

For more information, see Truncating Undo Tablespaces.

• [innodb\_merge\_threshold\_set\_all\_debug](#_bookmark202)

•

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-merge-threshold-set-all-  debug=# |
| System Variable | [innodb\_merge\_threshold\_set\_all\_debug](#_bookmark202) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50 |
| Minimum Value | 1 |
| Maximum Value | 50 |

Defines a page-full percentage value for index pages that overrides the current MERGE\_THRESHOLD setting for all indexes that are currently in the dictionary cache. This option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option. For related information, see [Section 15.8.11, “Configuring the Merge Threshold for Index Pages”](#_bookmark8) .

[innodb\_monitor\_disable](#_bookmark203)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-monitor-disable={counter |  module |pattern |all} |
| System Variable | [innodb\_monitor\_disable](#_bookmark203) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | String |

This variable acts as a switch, disabling InnoDB metrics counters. Counter data may be queried using the Information Schema INNODB\_METRICS table. For usage information, see [Section 15.15.6,](#_bookmark205) [“InnoDB INFORMATION\_SCHEMA Metrics Table”](#_bookmark205) .

[innodb\_monitor\_disable='latch'](#_bookmark203) disables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement” .

[innodb\_monitor\_enable](#_bookmark204)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-monitor-enable={counter |  module |pattern |all} |
| System Variable | [innodb\_monitor\_enable](#_bookmark204) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |

|  |  |
| --- | --- |
| Type | String |

This variable acts as a switch, enabling InnoDB metrics counters. Counter data may be queried using the Information Schema INNODB\_METRICS table. For usage information, see [Section 15.15.6,](#_bookmark205) [“InnoDB INFORMATION\_SCHEMA Metrics Table”](#_bookmark205) .

[innodb\_monitor\_enable='latch'](#_bookmark204) enables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement” .

• [innodb\_monitor\_reset](#_bookmark206)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-monitor-reset={counter |  module |pattern |all} |
| System Variable | [innodb\_monitor\_reset](#_bookmark206) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 15.15.6, “InnoDB INFORMATION\_SCHEMA Metrics Table”](#_bookmark205) .

[innodb\_monitor\_reset='latch'](#_bookmark206) resets statistics reported by SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement” .

[innodb\_monitor\_reset\_all](#_bookmark207)

•

|  |  |
| --- | --- |
| Command-Line Format | --innodb-monitor-reset-all={counter |  module |pattern |all} |
| System Variable | [innodb\_monitor\_reset\_all](#_bookmark207) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 15.15.6, “InnoDB INFORMATION\_SCHEMA Metrics](#_bookmark205) [Table”](#_bookmark205) .

• [innodb\_numa\_interleave](#_bookmark208)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-numa-interleave[={OFF |ON}] |
| System Variable | [innodb\_numa\_interleave](#_bookmark208) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enables the NUMA interleave memory policy for allocation of the InnoDB buffer pool. When [innodb\_numa\_interleave](#_bookmark208) 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](#_bookmark208) option to be available, MySQL must be compiled on a NUMA-enabled Linux system.

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

• [innodb\_old\_blocks\_pct](#_bookmark209)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-old-blocks-pct=# |
| System Variable | [innodb\_old\_blocks\_pct](#_bookmark209) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 37 |
| Minimum Value | 5 |
| Maximum Value | 95 |

Specifies the approximate percentage of the InnoDB buffer pool used for the old block sublist. The range of values is 5 to 95. The default value is 37 (that is, 3/8 of the pool). Often used in combination with [innodb\_old\_blocks\_time](#_bookmark210).

For more information, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant” . For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 15.5.1, “Buffer

Pool” .

• [innodb\_old\_blocks\_time](#_bookmark210)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-old-blocks-time=# |
| System Variable | [innodb\_old\_blocks\_time](#_bookmark210) |
| Scope | Global |



|  |  |
| --- | --- |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1000 |
| Minimum Value | 0 |
| Maximum Value | 2\*\*32-1 |
| Unit | milliseconds |

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

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

The default value is 1000.

This variable is often used in combination with [innodb\_old\_blocks\_pct](#_bookmark209). For more information, see Section 15.8.3.3, “Making the Buffer Pool Scan Resistant” . For information about buffer pool management, the LRU algorithm, and eviction policies, see Section 15.5.1, “Buffer Pool” .

• [innodb\_online\_alter\_log\_max\_size](#_bookmark71)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-online-alter-log-max-size=# |
| System Variable | [innodb\_online\_alter\_log\_max\_size](#_bookmark71) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 134217728 |
| Minimum Value | 65536 |
| Maximum Value | 2\*\*64-1 |
| Unit | bytes |

Specifies an upper limit in bytes on the size of the temporary log files used during online DDL operations for InnoDB tables. There is one such log file for each index being created or table being altered. This log file stores data inserted, updated, or deleted in the table during the DDL operation. The temporary log file is extended when needed by the value of [innodb\_sort\_buffer\_size](#_bookmark70), up to the maximum specified by [innodb\_online\_alter\_log\_max\_size](#_bookmark71). 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](#_bookmark211)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-open-files=# |

|  |  |
| --- | --- |
| System Variable | [innodb\_open\_files](#_bookmark211) |
| Scope | Global |
| Dynamic (≥ 8.0.28) | Yes |
| Dynamic (≤ 8.0.27) | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | -1 (signifies autosizing; do not assign this literal value) |
| Minimum Value | 10 |
| Maximum Value | 2147483647 |

Specifies the maximum number of files that InnoDB can have open at one time. The minimum value is 10. If [innodb\_file\_per\_table](#_bookmark17) is disabled, the default value is 300; otherwise, the default value is 300 or the table\_open\_cache setting, whichever is higher.

As of MySQL 8.0.28, the [innodb\_open\_files](#_bookmark211) limit can be set at runtime using a SELECT innodb\_set\_open\_files\_limit(*N*) statement, where *N* is the desired [innodb\_open\_files](#_bookmark211) limit; for example:

mysql> SELECT innodb\_set\_open\_files\_limit(1000);

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

It is not permitted to set [innodb\_open\_files](#_bookmark211) using a SET statement. To set [innodb\_open\_files](#_bookmark211) at runtime, use the SELECT innodb\_set\_open\_files\_limit(*N*) statement described above.

Setting [innodb\_open\_files=default](#_bookmark211) is not supported. Only integer values are permitted.

As of MySQL 8.0.28, to prevent non-LRU manged files from consuming the entire [innodb\_open\_files](#_bookmark211) limit, non-LRU managed files are limited to 90 percent of the [innodb\_open\_files](#_bookmark211) limit, which reserves 10 percent of the [innodb\_open\_files](#_bookmark211) limit for LRU managed files.

Temporary tablespace files were not counted toward the [innodb\_open\_files](#_bookmark211) limit from MySQL 8.0.24 to MySQL 8.0.27.

• [innodb\_optimize\_fulltext\_only](#_bookmark212)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-optimize-fulltext-  only[={OFF |ON}] |
| System Variable | [innodb\_optimize\_fulltext\_only](#_bookmark212) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

By default, OPTIMIZE TABLE reorganizes data in the clustered index of the table. When this option

added, deleted, and updated token data for InnoDB FULLTEXT indexes. For more information, see Optimizing InnoDB Full-Text Indexes.

• [innodb\_page\_cleaners](#_bookmark213)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-page-cleaners=# |
| System Variable | [innodb\_page\_cleaners](#_bookmark213) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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. 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](#_bookmark213) default value is 4. If the number of page cleaner threads exceeds the number of buffer pool instances, [innodb\_page\_cleaners](#_bookmark213) is automatically set to the same value as [innodb\_buffer\_pool\_instances](#_bookmark122).

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

Multithreaded page cleaner support extends to shutdown and recovery phases.

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

[Note] InnoDB: If the mysqld execution user is authorized, page cleaner

thread priority can be changed. See the man page of setpriority().

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

|  |  |  |  |
| --- | --- | --- | --- |
| mysql  mysql | hard  soft | nice  nice | -20  -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](#_bookmark7)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-page-size=# |
| System Variable | [innodb\_page\_size](#_bookmark7) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark7) 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 15.8.1, “InnoDB Startup Configuration” .

For both 32KB and 64KB page sizes, the maximum row length is approximately 16000 bytes. ROW\_FORMAT=COMPRESSED is not supported when [innodb\_page\_size](#_bookmark7) is set to 32KB or 64KB. For [innodb\_page\_size=32KB](#_bookmark7), extent size is 2MB. For [innodb\_page\_size=64KB](#_bookmark7), extent size is 4MB. [innodb\_log\_buffer\_size](#_bookmark185) 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](#_bookmark7) value. See the [innodb\_data\_file\_path](#_bookmark45) 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.

• [innodb\_parallel\_read\_threads](#_bookmark77)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-parallel-read-threads=# |
| Introduced | 8.0.14 |
| System Variable | [innodb\_parallel\_read\_threads](#_bookmark77) |
| Scope | Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 4 |
| Minimum Value | 1 |
| Maximum Value | 256 |

Defines the number of threads that can be used for parallel clustered index reads. Parallel scanning of partitions is supported as of MySQL 8.0.17. Parallel read threads can improve CHECK TABLE performance. InnoDB reads the clustered index twice during a CHECK TABLE operation. The second read can be performed in parallel. This feature does not apply to secondary index scans. The [innodb\_parallel\_read\_threads](#_bookmark77) session variable must be set to a value greater than 1 for parallel clustered index reads to occur. The actual number of threads used to perform a parallel clustered index read is determined by the [innodb\_parallel\_read\_threads](#_bookmark77) setting or the number of index subtrees to scan, whichever is smaller. The pages read into the buffer pool during the scan are kept at the tail of the buffer pool LRU list so that they can be discarded quickly when free buffer pool pages are required.

As of MySQL 8.0.17, the maximum number of parallel read threads (256) is the total number of threads for all client connections. If the thread limit is reached, connections fall back to using a single thread.

• [innodb\_print\_all\_deadlocks](#_bookmark214)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-print-all-deadlocks[={OFF |  ON}] |
| System Variable | [innodb\_print\_all\_deadlocks](#_bookmark214) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

When this option is enabled, information about all deadlocks in InnoDB user transactions is recorded in the mysqld error log. Otherwise, you see information about only the last deadlock, using the SHOW ENGINE INNODB STATUS 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 15.7.5, “Deadlocks in InnoDB” .

• [innodb\_print\_ddl\_logs](#_bookmark215)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-print-ddl-logs[={OFF |ON}] |
| System Variable | [innodb\_print\_ddl\_logs](#_bookmark215) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

• [innodb\_purge\_batch\_size](#_bookmark216)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-purge-batch-size=# |
| System Variable | [innodb\_purge\_batch\_size](#_bookmark216) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 300 |
| Minimum Value | 1 |
| Maximum Value | 5000 |

Defines the number of undo log pages that purge parses and processes in one batch from the history list. In a multithreaded purge configuration, the coordinator purge thread divides [innodb\_purge\_batch\_size](#_bookmark216) by [innodb\_purge\_threads](#_bookmark217) and assigns that number of pages to each purge thread. The [innodb\_purge\_batch\_size](#_bookmark216) 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](#_bookmark216) option is intended for advanced performance tuning in combination with the [innodb\_purge\_threads](#_bookmark217) setting. Most users need not change

[innodb\_purge\_batch\_size](#_bookmark216) from its default value.

For related information, see Section 15.8.9, “Purge Configuration” .

• [innodb\_purge\_threads](#_bookmark217)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-purge-threads=# |
| System Variable | [innodb\_purge\_threads](#_bookmark217) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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 15.8.9, “Purge Configuration” .

• [innodb\_purge\_rseg\_truncate\_frequency](#_bookmark218)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-purge-rseg-truncate-  frequency=# |
| System Variable | [innodb\_purge\_rseg\_truncate\_frequency](#_bookmark218) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 128 |
| Minimum Value | 1 |
| Maximum Value | 128 |

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

[innodb\_purge\_rseg\_truncate\_frequency](#_bookmark218) is intended for use with [innodb\_undo\_log\_truncate](#_bookmark201). For more information, see Truncating Undo Tablespaces.

• [innodb\_random\_read\_ahead](#_bookmark219)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-random-read-ahead[={OFF |  ON}] |
| System Variable | [innodb\_random\_read\_ahead](#_bookmark219) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

For details about performance considerations for different types of read-ahead requests, see Section 15.8.3.4, “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](#_bookmark220)

|  |  |  |
| --- | --- | --- |
|  | Command-Line Format | --innodb-read-ahead-threshold=# |
| System Variable | [innodb\_read\_ahead\_threshold](#_bookmark220) |
| 3308 | Scope | Global |



|  |  |
| --- | --- |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 56 |
| Minimum Value | 0 |
| Maximum Value | 64 |

Controls the sensitivity of linear read-ahead that InnoDB uses to prefetch pages into the buffer pool. If InnoDB reads at least [innodb\_read\_ahead\_threshold](#_bookmark220) 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](#_bookmark220) 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 15.8.3.4, “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](#_bookmark221)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-read-io-threads=# |
| System Variable | [innodb\_read\_io\_threads](#_bookmark221) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 4 |
| Minimum Value | 1 |
| Maximum Value | 64 |

The number of I/O threads for read operations in InnoDB. Its counterpart for write threads is [innodb\_write\_io\_threads](#_bookmark152). For more information, see Section 15.8.5, “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](#_bookmark221), [innodb\_write\_io\_threads](#_bookmark152), and the Linux aio-max-nr setting can



 workaround, you might reduce the settings for one or both of the MySQL

variables.

• [innodb\_read\_only](#_bookmark222)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-read-only[={OFF |ON}] |
| System Variable | [innodb\_read\_only](#_bookmark222) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Starts InnoDB in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. For more information, see Section 15.8.2, “Configuring InnoDB for Read-Only Operation” .

Previously, enabling the [innodb\_read\_only](#_bookmark222) system variable prevented creating and dropping tables only for the InnoDB storage engine. As of MySQL 8.0, enabling [innodb\_read\_only](#_bookmark222) prevents these operations for all storage engines. Table creation and drop operations for any storage engine modify data dictionary tables in the mysql system database, but those tables use the InnoDB storage engine and cannot be modified when [innodb\_read\_only](#_bookmark222) is enabled. The same principle applies to other table operations that require modifying data dictionary tables. Examples:

• If the [innodb\_read\_only](#_bookmark222) system variable is enabled, ANALYZE TABLE may fail because it cannot update statistics tables in the data dictionary, which use InnoDB. For ANALYZE TABLE operations that update the key distribution, failure may occur even if the operation updates the table itself (for example, if it is a MyISAM table). To obtain the updated distribution statistics, set information\_schema\_stats\_expiry=0.

• ALTER TABLE *tbl\_name* ENGINE=*engine\_name* fails because it updates the storage engine designation, which is stored in the data dictionary.

In addition, other tables in the mysql system database use the InnoDB storage engine in MySQL 8.0. Making those tables read only results in restrictions on operations that modify them. Examples:

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

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

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

• [innodb\_redo\_log\_archive\_dirs](#_bookmark223)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-redo-log-archive-dirs |
| Introduced | 8.0.17 |
| System Variable | [innodb\_redo\_log\_archive\_dirs](#_bookmark223) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | String |

|  |  |
| --- | --- |
| Default Value | NULL |

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

innodb\_redo\_log\_archive\_dirs='label1:/backups1;label2:/backups2'

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

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

• [innodb\_redo\_log\_capacity](#_bookmark15)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-redo-log-capacity=# |
| Introduced | 8.0.30 |
| System Variable | [innodb\_redo\_log\_capacity](#_bookmark15) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 104857600 |
| Minimum Value | 8388608 |
| Maximum Value | 137438953472 |
| Unit | bytes |

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

This variable supersedes the [innodb\_log\_files\_in\_group](#_bookmark11) and [innodb\_log\_file\_size](#_bookmark10) variables. When a innodb\_redo\_log\_capacity setting is defined, the innodb\_log\_files\_in\_group and innodb\_log\_file\_size settings are ignored;

otherwise, these settings are used to compute the [innodb\_redo\_log\_capacity](#_bookmark15) setting (innodb\_log\_files\_in\_group \* innodb\_log\_file\_size = innodb\_redo\_log\_capacity). If none of those variables are set, redo log capacity is set to the innodb\_redo\_log\_capacity default value.

For more information, see Section 15.6.5, “Redo Log” .

• [innodb\_redo\_log\_encrypt](#_bookmark95)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-redo-log-encrypt[={OFF |ON}] |
| System Variable | [innodb\_redo\_log\_encrypt](#_bookmark95) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Controls encryption of redo log data for tables encrypted using the InnoDB [data-at-rest encryption](#_bookmark66) [feature](#_bookmark66). Encryption of redo log data is disabled by default. For more information, see [Redo Log](#_bookmark85) [Encryption](#_bookmark85).

•

•

•

[innodb\_replication\_delay](#_bookmark224)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-replication-delay=# |
| System Variable | [innodb\_replication\_delay](#_bookmark224) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |
| Unit | milliseconds |

The replication thread delay in milliseconds on a replica server if [innodb\_thread\_concurrency](#_bookmark140) is reached.

[innodb\_rollback\_on\_timeout](#_bookmark184)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-rollback-on-timeout[={OFF |  ON}] |
| System Variable | [innodb\_rollback\_on\_timeout](#_bookmark184) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

InnoDB rolls back only the last statement on a transaction timeout by default. If [--innodb-](#_bookmark184) [rollback-on-timeout](#_bookmark184) is specified, a transaction timeout causes InnoDB to abort and roll back the entire transaction.

For more information, see Section 15.21.5, “InnoDB Error Handling” .

[innodb\_rollback\_segments](#_bookmark225)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-rollback-segments=# |
| System Variable | [innodb\_rollback\_segments](#_bookmark225) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 128 |
| Minimum Value | 1 |
| Maximum Value | 128 |

[innodb\_rollback\_segments](#_bookmark225) defines the number of rollback segments allocated to each undo tablespace and the global temporary tablespace for transactions that generate undo records. The number of transactions that each rollback segment supports depends on the InnoDB page size and the number of undo logs assigned to each transaction. For more information, see Section 15.6.6, “Undo Logs” .

•

•

•

For related information, see Section 15.3, “InnoDB Multi-Versioning” . For information about undo tablespaces, see Section 15.6.3.4, “Undo Tablespaces” .

[innodb\_saved\_page\_number\_debug](#_bookmark156)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-saved-page-number-debug=# |
| System Variable | [innodb\_saved\_page\_number\_debug](#_bookmark156) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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](#_bookmark155) option dirties the page defined by [innodb\_saved\_page\_number\_debug](#_bookmark156). The [innodb\_saved\_page\_number\_debug](#_bookmark156) option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

[innodb\_segment\_reserve\_factor](#_bookmark47)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-segment-reserve-factor=# |
| Introduced | 8.0.26 |
| System Variable | [innodb\_segment\_reserve\_factor](#_bookmark47) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Numeric |
| Default Value | 12.5 |
| Minimum Value | 0.03 |
| Maximum Value | 40 |

Defines the percentage of tablespace file segment pages reserved as empty pages. The setting is applicable to file-per-table and general tablespaces. The [innodb\_segment\_reserve\_factor](#_bookmark47) default setting is 12.5 percent, which is the same percentage of pages reserved in previous MySQL releases.

For more information, see [Configuring the Percentage of Reserved File Segment Pages](#_bookmark46).

[innodb\_sort\_buffer\_size](#_bookmark70)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-sort-buffer-size=# |
| System Variable | [innodb\_sort\_buffer\_size](#_bookmark70) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1048576 |
| Minimum Value | 65536 |

|  |  |
| --- | --- |
| Maximum Value | 67108864 |
| Unit | bytes |

This variable defines:

• The sort buffer size for online DDL operations that create or rebuild secondary indexes. However, as of MySQL 8.0.27, this responsibility is subsumed by the [innodb\_ddl\_buffer\_size](#_bookmark74) variable.

• 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 15.12.3, “Online DDL Space Requirements”](#_bookmark69) .

• [innodb\_spin\_wait\_delay](#_bookmark226)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-spin-wait-delay=# |
| System Variable | [innodb\_spin\_wait\_delay](#_bookmark226) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 6 |
| Minimum Value | 0 |
| Maximum Value (64-bit platforms, ≤ 8.0.13) | 2\*\*64-1 |
| Maximum Value (32-bit platforms, ≤ 8.0.13) | 2\*\*32-1 |
| Maximum Value (≥ 8.0.14) | 1000 |

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

Can be used in combination with the [innodb\_spin\_wait\_pause\_multiplier](#_bookmark227) variable for greater control over the duration of spin-lock polling delays.

For more information, see Section 15.8.8, “Configuring Spin Lock Polling” .

• [innodb\_spin\_wait\_pause\_multiplier](#_bookmark227)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-spin-wait-pause-  multiplier=# |
| Introduced | 8.0.16 |
| System Variable | [innodb\_spin\_wait\_pause\_multiplier](#_bookmark227) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 50 |
| Minimum Value | 1 |

|  |  |
| --- | --- |
| Maximum Value | 100 |

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

For more information, see Section 15.8.8, “Configuring Spin Lock Polling” .

• [innodb\_stats\_auto\_recalc](#_bookmark228)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-auto-recalc[={OFF |  ON}] |
| System Variable | [innodb\_stats\_auto\_recalc](#_bookmark228) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Causes InnoDB to automatically recalculate persistent statistics after the data in a table is changed substantially. The threshold value is 10% of the rows in the table. This setting applies to tables created when the [innodb\_stats\_persistent](#_bookmark2) 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](#_bookmark6) variable.

For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics

Parameters” .

• [innodb\_stats\_include\_delete\_marked](#_bookmark229)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-include-delete-  marked[={OFF |ON}] |
| System Variable | [innodb\_stats\_include\_delete\_marked](#_bookmark229) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

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

[innodb\_stats\_include\_delete\_marked](#_bookmark229) can be enabled to ensure that InnoDB includes delete-marked records when calculating persistent optimizer statistics.

When [innodb\_stats\_include\_delete\_marked](#_bookmark229) is enabled, ANALYZE TABLE considers delete- marked records when recalculating statistics.

[innodb\_stats\_include\_delete\_marked](#_bookmark229) is a global setting that affects all InnoDB tables. It is only applicable to persistent optimizer statistics.

For related information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics

Parameters” .

• [innodb\_stats\_method](#_bookmark230)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-method=value |
| System Variable | [innodb\_stats\_method](#_bookmark230) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 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 8.3.8, “InnoDB and MyISAM Index Statistics Collection” .

• [innodb\_stats\_on\_metadata](#_bookmark3)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-on-metadata[={OFF |  ON}] |
| System Variable | [innodb\_stats\_on\_metadata](#_bookmark3) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

This option only applies when optimizer statistics are configured to be non-persistent. Optimizer statistics are not persisted to disk when [innodb\_stats\_persistent](#_bookmark2) is disabled or when individual tables are created or altered with STATS\_PERSISTENT=0. For more information, see [Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”](#_bookmark1) .

When [innodb\_stats\_on\_metadata](#_bookmark3) is enabled, InnoDB updates non-persistent statistics when metadata statements such as SHOW TABLE STATUS or when accessing the Information Schema



TABLES or STATISTICS tables. (These updates are similar to what happens for ANALYZE TABLE.) When disabled, InnoDB does not update statistics during these operations. Leaving the setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve InnoDB tables.

To change the setting, issue the statement SET GLOBAL innodb\_stats\_on\_metadata=*mode*, where *mode* is either ON or OFF (or 1 or 0). Changing the setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”) and immediately affects the operation of all connections.

• [innodb\_stats\_persistent](#_bookmark2)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-persistent[={OFF |ON}] |
| System Variable | [innodb\_stats\_persistent](#_bookmark2) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Specifies whether InnoDB index statistics are persisted to disk. Otherwise, statistics may be recalculated frequently which can lead to variations in query execution plans. This setting is stored with each table when the table is created. You can set [innodb\_stats\_persistent](#_bookmark2) 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 15.8.10.1, “Configuring Persistent Optimizer Statistics

Parameters” .

• [innodb\_stats\_persistent\_sample\_pages](#_bookmark6)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-persistent-sample-  pages=# |
| System Variable | [innodb\_stats\_persistent\_sample\_pages](#_bookmark6) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 20 |
| Minimum Value | 1 |
| Maximum Value | 18446744073709551615 |

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of ANALYZE TABLE for an InnoDB table. For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters” .

**Note**

Setting a high value for [innodb\_stats\_persistent\_sample\_pages](#_bookmark6) could result in lengthy ANALYZE TABLE execution time. To estimate the number of database pages accessed by ANALYZE TABLE, see



[Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB](#_bookmark5) [Tables”](#_bookmark5) .

[innodb\_stats\_persistent\_sample\_pages](#_bookmark6) only applies when [innodb\_stats\_persistent](#_bookmark2) is enabled for a table; when [innodb\_stats\_persistent](#_bookmark2) is disabled, [innodb\_stats\_transient\_sample\_pages](#_bookmark4) applies instead.

• [innodb\_stats\_transient\_sample\_pages](#_bookmark4)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-stats-transient-sample-  pages=# |
| System Variable | [innodb\_stats\_transient\_sample\_pages](#_bookmark4) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 8 |
| Minimum Value | 1 |
| Maximum Value | 18446744073709551615 |

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an InnoDB table or recalculating statistics. For more information, see [Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”](#_bookmark1) .

**Note**

Setting a high value for [innodb\_stats\_transient\_sample\_pages](#_bookmark4) could result in lengthy ANALYZE TABLE execution time. To estimate the number of database pages accessed by ANALYZE TABLE, see [Section 15.8.10.3,](#_bookmark5) [“Estimating ANALYZE TABLE Complexity for InnoDB Tables”](#_bookmark5) .

[innodb\_stats\_transient\_sample\_pages](#_bookmark4) only applies when [innodb\_stats\_persistent](#_bookmark2) is disabled for a table; when [innodb\_stats\_persistent](#_bookmark2) is enabled, [innodb\_stats\_persistent\_sample\_pages](#_bookmark6) applies instead. Takes the place of [innodb\_stats\_sample\_pages](https://dev.mysql.com/doc/refman/5.7/en/innodb-parameters.html#sysvar_innodb_stats_sample_pages). For more information, see [Section 15.8.10.2, “Configuring Non-](#_bookmark1) [Persistent Optimizer Statistics Parameters”](#_bookmark1) .

• [innodb\_status\_output](#_bookmark231)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-status-output[={OFF |ON}] |
| System Variable | [innodb\_status\_output](#_bookmark231) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enables or disables periodic output for the standard InnoDB Monitor. Also used in combination with [innodb\_status\_output\_locks](#_bookmark232) to enable or disable periodic output for the InnoDB Lock Monitor. For more information, see [Section 15.17.2, “Enabling InnoDB Monitors”](#_bookmark101) .

[innodb\_status\_output\_locks](#_bookmark232)

•



|  |  |
| --- | --- |
| Command-Line Format | --innodb-status-output-locks[={OFF |  ON}] |
| System Variable | [innodb\_status\_output\_locks](#_bookmark232) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enables or disables the InnoDB Lock Monitor. When enabled, the InnoDB Lock Monitor prints additional information about locks in SHOW ENGINE INNODB STATUS output and in periodic output printed to the MySQL error log. Periodic output for the InnoDB Lock Monitor is printed as part of the standard InnoDB Monitor output. The standard InnoDB Monitor must therefore be enabled for the InnoDB Lock Monitor to print data to the MySQL error log periodically. For more information, see [Section 15.17.2, “Enabling InnoDB Monitors”](#_bookmark101) .

• [innodb\_strict\_mode](#_bookmark20)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-strict-mode[={OFF |ON}] |
| System Variable | [innodb\_strict\_mode](#_bookmark20) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

When [innodb\_strict\_mode](#_bookmark20) 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](#_bookmark20) on the command line when starting mysqld, or in a MySQL configuration file. You can also enable or disable [innodb\_strict\_mode](#_bookmark20) 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 5.1.9.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the SESSION setting for [innodb\_strict\_mode](#_bookmark20), and the setting affects only that client.

As of MySQL 8.0.26, setting the session value of this system variable is a restricted operation. The session user must have privileges sufficient to set restricted session variables. See Section 5.1.9.1, “System Variable Privileges” .

• [innodb\_sync\_array\_size](#_bookmark233)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-sync-array-size=# |
| System Variable | [innodb\_sync\_array\_size](#_bookmark233) |
| Scope | Global |

|  |  |
| --- | --- |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 1 |
| Minimum Value | 1 |
| Maximum Value | 1024 |

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

• [innodb\_sync\_spin\_loops](#_bookmark234)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-sync-spin-loops=# |
| System Variable | [innodb\_sync\_spin\_loops](#_bookmark234) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 30 |
| Minimum Value | 0 |
| Maximum Value | 4294967295 |

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

• [innodb\_sync\_debug](#_bookmark235)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-sync-debug[={OFF |ON}] |
| System Variable | [innodb\_sync\_debug](#_bookmark235) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Enables sync debug checking for the InnoDB storage engine. This option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

• [innodb\_table\_locks](#_bookmark236)

|  |  |  |
| --- | --- | --- |
|  | Command-Line Format | --innodb-table-locks[={OFF |ON}] |
| System Variable | [innodb\_table\_locks](#_bookmark236) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| 3320 | 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](#_bookmark236) is 1, which means that LOCK TABLES causes InnoDB to lock a table internally if autocommit = 0.

[innodb\_table\_locks = 0](#_bookmark236) 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 15.7, “InnoDB Locking and Transaction Model” .

• [innodb\_temp\_data\_file\_path](#_bookmark237)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-temp-data-file-  path=file\_name |
| System Variable | [innodb\_temp\_data\_file\_path](#_bookmark237) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | String |
| Default Value | ibtmp1:12M:autoextend |

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

If no value is specified for [innodb\_temp\_data\_file\_path](#_bookmark237), the default behavior is to create a single auto-extending data file named ibtmp1 in the [innodb\_data\_home\_dir](#_bookmark142) directory. The initial file size is slightly larger than 12MB.

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

*file\_name*:*file\_size* [:autoextend[:max:*max\_file\_size*]]

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

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

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

The autoextend and max attributes can be used only for the data file specified last in the [innodb\_temp\_data\_file\_path](#_bookmark237) setting. For example:

[mysqld]

innodb\_temp\_data\_file\_path=ibtmp1:50M;ibtmp2:12M:autoextend:max:500M

The autoextend option causes the data file to automatically increase in size when it runs out of free space. The autoextend increment is 64MB by default. To modify the increment, change the [innodb\_autoextend\_increment](#_bookmark118) variable setting.

The directory path for global temporary tablespace data files is formed by concatenating the paths defined by [innodb\_data\_home\_dir](#_bookmark142) and [innodb\_temp\_data\_file\_path](#_bookmark237).

Before running InnoDB in read-only mode, set [innodb\_temp\_data\_file\_path](#_bookmark237) to a location outside of the data directory. The path must be relative to the data directory. For example:

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

For more information, see Global Temporary Tablespace.

• [innodb\_temp\_tablespaces\_dir](#_bookmark238)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-temp-tablespaces-  dir=dir\_name |
| Introduced | 8.0.13 |
| System Variable | [innodb\_temp\_tablespaces\_dir](#_bookmark238) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |
| Default Value | #innodb\_temp |

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

As of MySQL 8.0. 16, session temporary tablespaces always store user-created temporary tables and internal temporary tables created by the optimizer using InnoDB.

(Previously, the on-disk storage engine for internal temporary tables was determined by the internal\_tmp\_disk\_storage\_engine system variable, which is no longer supported. See Storage Engine for On-Disk Internal Temporary Tables.)

For more information, see Session Temporary Tablespaces.

• [innodb\_thread\_concurrency](#_bookmark140)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-thread-concurrency=# |
| System Variable | [innodb\_thread\_concurrency](#_bookmark140) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |



|  |  |
| --- | --- |
| Maximum Value | 1000 |

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

InnoDB tries to keep the number of threads inside InnoDB less than or equal to the [innodb\_thread\_concurrency](#_bookmark140) 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](#_bookmark140) 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](#_bookmark140) 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](#_bookmark140) (no limit).

• If your workload is consistently heavy or occasionally spikes, set an [innodb\_thread\_concurrency](#_bookmark140) 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](#_bookmark140) 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](#_bookmark140) 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](#_bookmark140) setting can be smaller than the number of virtual CPUs.

• An [innodb\_thread\_concurrency](#_bookmark140) 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](#_bookmark140) 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 15.8.4, “Configuring Thread Concurrency for InnoDB” .

• [innodb\_thread\_sleep\_delay](#_bookmark112)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-thread-sleep-delay=# |
| System Variable | [innodb\_thread\_sleep\_delay](#_bookmark112) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 10000 |
| Minimum Value | 0 |
| Maximum Value | 1000000 |
| Unit | microseconds |

How long InnoDB threads sleep before joining the InnoDB queue, in microseconds. The default value is 10000. A value of 0 disables sleep. You can set [innodb\_adaptive\_max\_sleep\_delay](#_bookmark111) to the highest value you would allow for [innodb\_thread\_sleep\_delay](#_bookmark112), and InnoDB automatically adjusts [innodb\_thread\_sleep\_delay](#_bookmark112) 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 15.8.4, “Configuring Thread Concurrency for InnoDB” .

• [innodb\_tmpdir](#_bookmark72)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-tmpdir=dir\_name |
| System Variable | [innodb\_tmpdir](#_bookmark72) |
| Scope | Global, Session |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Directory name |
| Default Value | NULL |

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

Online ALTER TABLE operations that rebuild the table also create an *intermediate* table file in the same directory as the original table. The [innodb\_tmpdir](#_bookmark72) 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](#_bookmark72) 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



debug

TABLE operation reports an error if [innodb\_tmpdir](#_bookmark72) is set to an invalid directory. [innodb\_tmpdir](#_bookmark72)

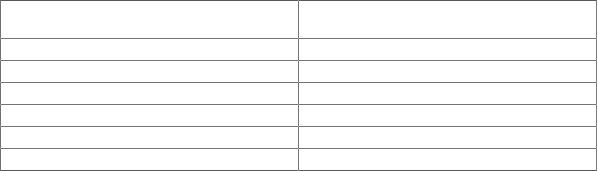
overrides the MySQL tmpdir setting but only for online ALTER TABLE operations. The FILE privilege is required to configure [innodb\_tmpdir](#_bookmark72).

The [innodb\_tmpdir](#_bookmark72) 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](#_bookmark72) setting if all servers have the same operating system environment. Otherwise, replicating the [innodb\_tmpdir](#_bookmark72) 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](#_bookmark72) on each server individually.

For more information, see [Section 15.12.3, “Online DDL Space Requirements”](#_bookmark69) . For information about online ALTER TABLE operations, see [Section 15.12, “InnoDB and Online DDL”](#_bookmark48) .

• [innodb\_trx\_purge\_view\_update\_only\_debug](#_bookmark239)



--innodb-trx-purge-view-update-only-

debug[={OFF |ON}]

[innodb\_trx\_purge\_view\_update\_only\_](#_bookmark239)

Global

Yes

No

Boolean

OFF

Command-Line Format

System Variable

Scope

Dynamic

SET\_VAR Hint Applies Type

Default Value

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](#_bookmark240)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-trx-rseg-n-slots-debug=# |
| System Variable | [innodb\_trx\_rseg\_n\_slots\_debug](#_bookmark240) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 0 |
| Minimum Value | 0 |
| Maximum Value | 1024 |

Sets a debug flag that limits TRX\_RSEG\_N\_SLOTS to a given value for the trx\_rsegf\_undo\_find\_free function that looks for free slots for undo log segments. This option is only available if debugging support is compiled in using the WITH\_DEBUG CMake option.

• [innodb\_undo\_directory](#_bookmark146)

|  |  |
| --- | --- |
| System Variable | [innodb\_undo\_directory](#_bookmark146) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Directory name |

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

There is no default value (it is NULL). If the [innodb\_undo\_directory](#_bookmark146) variable is undefined, undo tablespaces are created in the data directory.

The default undo tablespaces (innodb\_undo\_001 and innodb\_undo\_002) created when the MySQL instance is initialized always reside in the directory defined by the [innodb\_undo\_directory](#_bookmark146) variable.

Undo tablespaces created using CREATE UNDO TABLESPACE syntax are created in the directory

defined by the [innodb\_undo\_directory](#_bookmark146) variable if a different path is not specified. For more information, see Section 15.6.3.4, “Undo Tablespaces” .

• [innodb\_undo\_log\_encrypt](#_bookmark96)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-undo-log-encrypt[={OFF |ON}] |
| System Variable | [innodb\_undo\_log\_encrypt](#_bookmark96) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

Controls encryption of undo log data for tables encrypted using the InnoDB [data-at-rest encryption](#_bookmark66) [feature](#_bookmark66). Only applies to undo logs that reside in separate undo tablespaces. See Section 15.6.3.4, “Undo Tablespaces” . Encryption is not supported for undo log data that resides in the system tablespace. For more information, see [Undo Log Encryption](#_bookmark86).

• [innodb\_undo\_log\_truncate](#_bookmark201)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-undo-log-truncate[={OFF |  ON}] |
| System Variable | [innodb\_undo\_log\_truncate](#_bookmark201) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

When enabled, undo tablespaces that exceed the threshold value defined by [innodb\_max\_undo\_log\_size](#_bookmark200) are marked for truncation. Only undo tablespaces can be truncated.



Truncating undo logs that reside in the system tablespace is not supported. For truncation to occur, there must be at least two undo tablespaces.

The [innodb\_purge\_rseg\_truncate\_frequency](#_bookmark218) variable can be used to expedite truncation of undo tablespaces.

For more information, see Truncating Undo Tablespaces.

• [innodb\_undo\_tablespaces](#_bookmark241)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-undo-tablespaces=# |
| Deprecated | Yes |
| System Variable | [innodb\_undo\_tablespaces](#_bookmark241) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 2 |
| Minimum Value | 2 |
| Maximum Value | 127 |

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

**Note**

The [innodb\_undo\_tablespaces](#_bookmark241) variable is deprecated and is no longer configurable as of MySQL 8.0.14. Expect it to be removed in a future release.

For more information, see Section 15.6.3.4, “Undo Tablespaces” .

• [innodb\_use\_fdatasync](#_bookmark160)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-use-fdatasync[={OFF |ON}] |
| Introduced | 8.0.26 |
| System Variable | [innodb\_use\_fdatasync](#_bookmark160) |
| Scope | Global |
| Dynamic | Yes |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | OFF |

On platforms that support fdatasync() system calls, enabling the [innodb\_use\_fdatasync](#_bookmark160) variable permits using fdatasync() instead of fsync() system calls for operating system flushes. An fdatasync() call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

A subset of [innodb\_flush\_method](#_bookmark12) settings such as fsync, O\_DSYNC, and O\_DIRECT use fsync() system calls. The [innodb\_use\_fdatasync](#_bookmark160) variable is applicable when using those settings.

• [innodb\_use\_native\_aio](#_bookmark242)

|  |  |
| --- | --- |
| System Variable | [innodb\_use\_native\_aio](#_bookmark242) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | 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](#_bookmark242). 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 15.8.6, “Using Asynchronous I/O on Linux” .

• innodb\_validate\_tablespace\_paths

|  |  |
| --- | --- |
| Command-Line Format | --innodb-validate-tablespace-  paths[={OFF |ON}] |
| Introduced | 8.0.21 |
| System Variable | [innodb\_validate\_tablespace\_paths](#_bookmark243) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Boolean |
| Default Value | ON |

Controls tablespace file path validation. At startup, InnoDB validates the paths of known tablespace files against tablespace file paths stored in the data dictionary in case tablespace files have been moved to a different location. The [innodb\_validate\_tablespace\_paths](#_bookmark243) variable permits disabling tablespace path validation. This feature is intended for environments where tablespaces



files are not moved. Disabling path validation improves startup time on systems with a large number of tablespace files.

**Warning**

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

For more information, see Section 15.6.3.7, “Disabling Tablespace Path Validation” .

• [innodb\_version](#_bookmark100)

The InnoDB version number. In MySQL 8.0, separate version numbering for InnoDB does not apply and this value is the same the version number of the server.

• [innodb\_write\_io\_threads](#_bookmark152)

|  |  |
| --- | --- |
| Command-Line Format | --innodb-write-io-threads=# |
| System Variable | [innodb\_write\_io\_threads](#_bookmark152) |
| Scope | Global |
| Dynamic | No |
| SET\_VAR Hint Applies | No |
| Type | Integer |
| Default Value | 4 |
| Minimum Value | 1 |
| Maximum Value | 64 |

The number of I/O threads for write operations in InnoDB. The default value is 4. Its counterpart for read threads is [innodb\_read\_io\_threads](#_bookmark221). For more information, see Section 15.8.5, “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](#_bookmark221), [innodb\_write\_io\_threads](#_bookmark152), 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” .

**15.15** **InnoDB** **INFORMATION\_SCHEMA** **Tables**

This section provides information and usage examples for InnoDB INFORMATION\_SCHEMA tables.

InnoDB INFORMATION\_SCHEMA tables provide metadata, status information, and statistics about various aspects of the InnoDB storage engine. You can view a list of InnoDB INFORMATION\_SCHEMA tables by issuing a SHOW TABLES statement on the INFORMATION\_SCHEMA database:

mysql> **SHOW** **TABLES** **FROM** **INFORMATION\_SCHEMA** **LIKE** **'INNODB%';**

For table definitions, see Section 26.4, “INFORMATION\_SCHEMA InnoDB Tables” . For general information regarding the MySQL INFORMATION\_SCHEMA database, see Chapter 26, *INFORMATION\_SCHEMA* *Tables*.

**15.15.1** **InnoDB** **INFORMATION\_SCHEMA** **Tables** **about** **Compression**

There are two pairs of InnoDB INFORMATION\_SCHEMA tables about compression that can provide insight into how well compression is working overall:

• INNODB\_CMP and INNODB\_CMP\_RESET provide information about the number of compression operations and the amount of time spent performing compression.

• INNODB\_CMPMEM and INNODB\_CMPMEM\_RESET provide information about the way memory is allocated for compression.

**15.15.1.1** **INNODB\_CMP** **and** **INNODB\_CMP\_RESET**

The INNODB\_CMP and INNODB\_CMP\_RESET tables provide status information about operations related to compressed tables, which are described in [Section 15.9, “InnoDB Table and Page Compression”](#_bookmark16) . The PAGE\_SIZE column reports the compressed page size.

These two tables have identical contents, but reading from INNODB\_CMP\_RESET resets the statistics on compression and uncompression operations. For example, if you archive the output of INNODB\_CMP\_RESET every 60 minutes, you see the statistics for each hourly period. If you monitor the output of INNODB\_CMP (making sure never to read INNODB\_CMP\_RESET), you see the cumulative statistics since InnoDB was started.

For the table definition, see Section 26.4.6, “The INFORMATION\_SCHEMA INNODB\_CMP and INNODB\_CMP\_RESET Tables” .

**15.15.1.2** **INNODB\_CMPMEM** **and** **INNODB\_CMPMEM\_RESET**

The INNODB\_CMPMEM and INNODB\_CMPMEM\_RESET tables provide status information about compressed pages that reside in the buffer pool. Please consult [Section 15.9, “InnoDB Table and](#_bookmark16) [Page Compression”](#_bookmark16) for further information on compressed tables and the use of the buffer pool. The INNODB\_CMP and INNODB\_CMP\_RESET tables should provide more useful statistics on compression.

**Internal** **Details**

InnoDB uses a buddy allocator system to manage memory allocated to pages of various sizes, from 1KB to 16KB. Each row of the two tables described here corresponds to a single page size.

The INNODB\_CMPMEM and INNODB\_CMPMEM\_RESET tables have identical contents, but reading from INNODB\_CMPMEM\_RESET resets the statistics on relocation operations. For example, if every 60 minutes you archived the output of INNODB\_CMPMEM\_RESET, it would show the hourly statistics. If you never read INNODB\_CMPMEM\_RESET and monitored the output of INNODB\_CMPMEM instead, it would show the cumulative statistics since InnoDB was started.

For the table definition, see Section 26.4.7, “The INFORMATION\_SCHEMA INNODB\_CMPMEM and INNODB\_CMPMEM\_RESET Tables” .

**15.15.1.3** **Using** **the** **Compression** **Information** **Schema** **Tables**

**Example** **15.1** **Using** **the** **Compression** **Information** **Schema** **Tables**

The following is sample output from a database that contains compressed tables (see [Section 15.9,](#_bookmark16) [“InnoDB Table and Page Compression”](#_bookmark16) , INNODB\_CMP, INNODB\_CMP\_PER\_INDEX, and INNODB\_CMPMEM).



The following table shows the contents of INFORMATION\_SCHEMA.INNODB\_CMP under a light workload. The only compressed page size that the buffer pool contains is 8K. Compressing or uncompressing pages has consumed less than a second since the time the statistics were reset, because the columns COMPRESS\_TIME and UNCOMPRESS\_TIME are zero.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **page** **size** | **compress** **ops** | **compress** **ops** **ok** | **compress** **time** | **uncompress** **ops** | **uncompress** **time** |
| 1024 | 0 | 0 | 0 | 0 | 0 |
| 2048 | 0 | 0 | 0 | 0 | 0 |
| 4096 | 0 | 0 | 0 | 0 | 0 |
| 8192 | 1048 | 921 | 0 | 61 | 0 |
| 16384 | 0 | 0 | 0 | 0 | 0 |

According to INNODB\_CMPMEM, there are 6169 compressed 8KB pages in the buffer pool. The only other allocated block size is 64 bytes. The smallest PAGE\_SIZE in INNODB\_CMPMEM is used for block descriptors of those compressed pages for which no uncompressed page exists in the buffer pool. We see that there are 5910 such pages. Indirectly, we see that 259 (6169-5910) compressed pages also exist in the buffer pool in uncompressed form.

The following table shows the contents of INFORMATION\_SCHEMA.INNODB\_CMPMEM under a light workload. Some memory is unusable due to fragmentation of the memory allocator for compressed pages: SUM(PAGE\_SIZE\*PAGES\_FREE)=6784. This is because small memory allocation requests are fulfilled by splitting bigger blocks, starting from the 16K blocks that are allocated from the main buffer pool, using the buddy allocation system. The fragmentation is this low because some allocated blocks have been relocated (copied) to form bigger adjacent free blocks. This copying of SUM(PAGE\_SIZE\*RELOCATION\_OPS) bytes has consumed less than a second

(SUM(RELOCATION\_TIME)=0).

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **page** **size** | **pages** **used** | **pages** **free** | **relocation** **ops** | **relocation** **time** |
| 64 | 5910 | 0 | 2436 | 0 |
| 128 | 0 | 1 | 0 | 0 |
| 256 | 0 | 0 | 0 | 0 |
| 512 | 0 | 1 | 0 | 0 |
| 1024 | 0 | 0 | 0 | 0 |
| 2048 | 0 | 1 | 0 | 0 |
| 4096 | 0 | 1 | 0 | 0 |
| 8192 | 6169 | 0 | 5 | 0 |
| 16384 | 0 | 0 | 0 | 0 |

**15.15.2** **InnoDB** **INFORMATION\_SCHEMA** **Transaction** **and** **Locking**

**Information**

**Note**

This section describes locking information as exposed by the Performance Schema data\_locks and data\_lock\_waits tables, which supersede the INFORMATION\_SCHEMA INNODB\_LOCKS and INNODB\_LOCK\_WAITS tables in MySQL 8.0. For similar discussion written in terms of the older INFORMATION\_SCHEMA tables, see [InnoDB INFORMATION\_SCHEMA](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-transactions.html) [Transaction and Locking Information](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-transactions.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

One INFORMATION\_SCHEMA table and two Performance Schema tables enable you to monitor InnoDB transactions and diagnose potential locking problems:



• INNODB\_TRX: This INFORMATION\_SCHEMA table provides information about every transaction currently executing inside InnoDB, including the transaction state (for example, whether it is running or waiting for a lock), when the transaction started, and the particular SQL statement the transaction is executing.

• data\_locks: This Performance Schema table contains a row for each hold lock and each lock request that is blocked waiting for a held lock to be released:

• There is one row for each held lock, whatever the state of the transaction that holds the lock ( INNODB\_TRX.TRX\_STATE is RUNNING, LOCK WAIT, ROLLING BACK or COMMITTING).

• Each transaction in InnoDB that is waiting for another transaction to release a lock ( INNODB\_TRX.TRX\_STATE is LOCK WAIT) is blocked by exactly one blocking lock request. That blocking lock request is for a row or table lock held by another transaction in an incompatible mode. A lock request always has a mode that is incompatible with the mode of the held lock that blocks the request (read vs. write, shared vs. exclusive).

The blocked transaction cannot proceed until the other transaction commits or rolls back, thereby releasing the requested lock. For every blocked transaction, data\_locks contains one row that describes each lock the transaction has requested, and for which it is waiting.

• data\_lock\_waits: This Performance Schema table indicates which transactions are waiting for a given lock, or for which lock a given transaction is waiting. This table contains one or more rows for each blocked transaction, indicating the lock it has requested and any locks that are blocking that request. The REQUESTING\_ENGINE\_LOCK\_ID value refers to the lock requested by a transaction, and the BLOCKING\_ENGINE\_LOCK\_ID value refers to the lock (held by another transaction)

that prevents the first transaction from proceeding. For any given blocked transaction, all rows in data\_lock\_waits have the same value for REQUESTING\_ENGINE\_LOCK\_ID and different values for BLOCKING\_ENGINE\_LOCK\_ID.

For more information about the preceding tables, see Section 26.4.28, “The INFORMATION\_SCHEMA INNODB\_TRX Table” , Section 27.12.13.1, “The data\_locks Table” , and Section 27.12.13.2, “The data\_lock\_waits Table” .

**15.15.2.1** **Using** **InnoDB** **Transaction** **and** **Locking** **Information**

**Note**

This section describes locking information as exposed by the Performance Schema data\_locks and data\_lock\_waits tables, which supersede the INFORMATION\_SCHEMA INNODB\_LOCKS and INNODB\_LOCK\_WAITS tables in MySQL 8.0. For similar discussion written in terms of the older INFORMATION\_SCHEMA tables, see [Using InnoDB Transaction and Locking](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-examples.html) [Information](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-examples.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

**Identifying** **Blocking** **Transactions**

It is sometimes helpful to identify which transaction blocks another. The tables that contain information about InnoDB transactions and data locks enable you to determine which transaction is waiting for another, and which resource is being requested. (For descriptions of these tables, see [Section 15.15.2,](#_bookmark244) [“InnoDB INFORMATION\_SCHEMA Transaction and Locking Information”](#_bookmark244) .)

Suppose that three sessions are running concurrently. Each session corresponds to a MySQL thread, and executes one transaction after another. Consider the state of the system when these sessions have issued the following statements, but none has yet committed its transaction:

• Session A:

BEGIN;

SELECT a FROM t FOR UPDATE;

SELECT SLEEP(100);

• Session B:

SELECT b FROM t FOR UPDATE;

• Session C:

SELECT c FROM t FOR UPDATE;

In this scenario, use the following query to see which transactions are waiting and which transactions are blocking them:

SELECT

r.trx\_id waiting\_trx\_id,

r .trx\_mysql\_thread\_id waiting\_thread,

r .trx\_query waiting\_query,

b .trx\_id blocking\_trx\_id,

b .trx\_mysql\_thread\_id blocking\_thread,

b .trx\_query blocking\_query

FROM performance\_schema .data\_lock\_waits w

INNER JOIN information\_schema .innodb\_trx b

ON b .trx\_id = w .blocking\_engine\_transaction\_id

INNER JOIN information\_schema .innodb\_trx r

ON r.trx\_id = w.requesting\_engine\_transaction\_id;

Or, more simply, use the sys schema innodb\_lock\_waits view:

SELECT

waiting\_trx\_id,

waiting\_pid,

waiting\_query,

blocking\_trx\_id,

blocking\_pid,

blocking\_query

FROM sys.innodb\_lock\_waits;

If a NULL value is reported for the blocking query, see [Identifying a Blocking Query After the Issuing](#_bookmark245) [Session Becomes Idle](#_bookmark245).

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| **waiting** **trx** **id** | **waiting** **thread** | **waiting** **query** | **blocking** **trx** **id** | **blocking**  **thread** | **blocking** **query** |
| A4 | 6 | SELECT b  FROM t FOR  UPDATE | A3 | 5 | SELECT  SLEEP(100) |
| A5 | 7 | SELECT c  FROM t FOR  UPDATE | A3 | 5 | SELECT  SLEEP(100) |
| A5 | 7 | SELECT c  FROM t FOR  UPDATE | A4 | 6 | SELECT b  FROM t FOR  UPDATE |

In the preceding table, you can identify sessions by the “waiting query” or “blocking query” columns. As you can see:

• Session B (trx id A4, thread 6) and Session C (trx id A5, thread 7) are both waiting for Session A (trx id A3, thread 5).

• Session C is waiting for Session B as well as Session A.

You can see the underlying data in the INFORMATION\_SCHEMA INNODB\_TRX table and Performance Schema data\_locks and data\_lock\_waits tables.

The following table shows some sample contents of the INNODB\_TRX table.

)

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **trx** **id** | **trx** **state** | **trx** **started** | **trx** **requested** **lock** **id** | **trx** **wait**  **started** | **trx** **weight** | **trx** **mysql** **thread** **id** | **trx** **query** |
| A3 | RUNNING | 2008-01-1  16:44:54 | 5NULL | NULL | 2 | 5 | SELECT  SLEEP(100 |
| A4 | LOCK  WAIT | 2008-01-1  16:45:09 | 5A4:1:3:2 | 2008-01-1  16:45:09 | 52 | 6 | SELECT  b FROM  t FOR  UPDATE |
| A5 | LOCK  WAIT | 2008-01-1  16:45:14 | 5A5:1:3:2 | 2008-01-1  16:45:14 | 52 | 7 | SELECT  c FROM  t FOR  UPDATE |

The following table shows some sample contents of the data\_locks table.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **lock** **id** | **lock** **trx** **id** | **lock** **mode** | **lock** **type** | **lock**  **schema** | **lock** **table** | **lock** **index** | **lock** **data** |
| A3:1:3:2 | A3 | X | RECORD | test | t | PRIMARY | 0x0200 |
| A4:1:3:2 | A4 | X | RECORD | test | t | PRIMARY | 0x0200 |
| A5:1:3:2 | A5 | X | RECORD | test | t | PRIMARY | 0x0200 |

The following table shows some sample contents of the data\_lock\_waits table.

|  |  |  |  |
| --- | --- | --- | --- |
| **requesting** **trx** **id** | **requested** **lock** **id** | **blocking** **trx** **id** | **blocking** **lock** **id** |
| A4 | A4:1:3:2 | A3 | A3:1:3:2 |
| A5 | A5:1:3:2 | A3 | A3:1:3:2 |
| A5 | A5:1:3:2 | A4 | A4:1:3:2 |

**Identifying** **a** **Blocking** **Query** **After** **the** **Issuing** **Session** **Becomes** **Idle**

When identifying blocking transactions, a NULL value is reported for the blocking query if the session that issued the query has become idle. In this case, use the following steps to determine the blocking query:

1. Identify the processlist ID of the blocking transaction. In the sys.innodb\_lock\_waits table, the processlist ID of the blocking transaction is the blocking\_pid value.

2. Using the blocking\_pid, query the MySQL Performance Schema threads table to determine the THREAD\_ID of the blocking transaction. For example, if the blocking\_pid is 6, issue this query:

SELECT THREAD\_ID FROM performance\_schema.threads WHERE PROCESSLIST\_ID = 6;

3. Using the THREAD\_ID, query the Performance Schema events\_statements\_current table to determine the last query executed by the thread. For example, if the THREAD\_ID is 28, issue this query:

SELECT THREAD\_ID, SQL\_TEXT FROM performance\_schema.events\_statements\_current

WHERE THREAD\_ID = 28\G

4. If the last query executed by the thread is not enough information to determine why a lock is held, you can query the Performance Schema events\_statements\_history table to view the last 10 statements executed by the thread.

SELECT THREAD\_ID, SQL\_TEXT FROM performance\_schema.events\_statements\_history

WHERE THREAD\_ID = 28 ORDER BY EVENT\_ID;



PROCESSLIST

**Correlating** **InnoDB** **Transactions** **with** **MySQL** **Sessions**

Sometimes it is useful to correlate internal InnoDB locking information with the session-level information maintained by MySQL. For example, you might like to know, for a given InnoDB transaction ID, the corresponding MySQL session ID and name of the session that may be holding a lock, and thus blocking other transactions.

The following output from the INFORMATION\_SCHEMA INNODB\_TRX table and Performance Schema data\_locks and data\_lock\_waits tables is taken from a somewhat loaded system. As can be seen, there are several transactions running.

The following data\_locks and data\_lock\_waits tables show that:

• Transaction 77F (executing an INSERT) is waiting for transactions 77E, 77D, and 77B to commit.

• Transaction 77E (executing an INSERT) is waiting for transactions 77D and 77B to commit.

• Transaction 77D (executing an INSERT) is waiting for transaction 77B to commit.

• Transaction 77B (executing an INSERT) is waiting for transaction 77A to commit.

• Transaction 77A is running, currently executing SELECT.

• Transaction E56 (executing an INSERT) is waiting for transaction E55 to commit.

• Transaction E55 (executing an INSERT) is waiting for transaction 19C to commit.

• Transaction 19C is running, currently executing an INSERT.

**Note**

There may be inconsistencies between queries shown in the INFORMATION\_SCHEMA PROCESSLIST and INNODB\_TRX tables. For an explanation, see [Section 15.15.2.3, “Persistence and Consistency of InnoDB](#_bookmark246) [Transaction and Locking Information”](#_bookmark246) .

The following table shows the contents of the PROCESSLIST table for a system running a heavy workload.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **ID** | **USER** | **HOST** | **DB** | **COMMAND** | **TIME** | **STATE** | **INFO** |
| 384 | root | localhost | test | Query | 10 | update | INSERT INTO t2 VALUES … |
| 257 | root | localhost | test | Query | 3 | update | INSERT INTO t2 VALUES … |
| 130 | root | localhost | test | Query | 0 | update | INSERT INTO t2 VALUES … |
| 61 | root | localhost | test | Query | 1 | update | INSERT INTO t2 VALUES … |
| 8 | root | localhost | test | Query | 1 | update | INSERT INTO t2 VALUES … |
| 4 | root | localhost | test | Query | 0 | preparing | SELECT  \* FROM |
| 2 | root | localhost | test | Sleep | 566 |  | NULL |

The following table shows the contents of the INNODB\_TRX table for a system running a heavy workload.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **trx** **id** | **trx** **state** | **trx** **started** | **trx** **requested** **lock** **id** | **trx** **wait**  **started** | **trx** **weight** | **trx** **mysql** **thread** **id** | **trx** **query** |
| 77F | LOCK  WAIT | 2008-01-1  13:10:16 | 577F | 2008-01-1  13:10:16 | 51 | 876 | INSERT INTO t09 (D, B, C) VALUES … |
| 77E | LOCK  WAIT | 2008-01-1  13:10:16 | 577E | 2008-01-1  13:10:16 | 51 | 875 | INSERT INTO t09 (D, B, C) VALUES … |
| 77D | LOCK  WAIT | 2008-01-1  13:10:16 | 577D | 2008-01-1  13:10:16 | 51 | 874 | INSERT INTO t09 (D, B, C) VALUES … |
| 77B | LOCK  WAIT | 2008-01-1  13:10:16 | 577B:733:1 | **2**:0108-01-1  13:10:16 | 54 | 873 | INSERT INTO t09 (D, B, C) VALUES … |
| 77A | RUNNING | 2008-01-1  13:10:16 | 5NULL | NULL | 4 | 872 | SELECT  b, c  FROM t09  WHERE … |
| E56 | LOCK  WAIT | 2008-01-1  13:10:06 | 5E56:743:6 | :**2**008-01-1  13:10:06 | **5** | 384 | INSERT INTO t2 VALUES … |
| E55 | LOCK  WAIT | 2008-01-1  13:10:06 | 5E55:743:3 | 82:0208-01-1  13:10:13 | 5965 | 257 | INSERT INTO t2 VALUES … |
| 19C | RUNNING | 2008-01-1  13:09:10 | 5NULL | NULL | 2900 | 130 | INSERT INTO t2 VALUES … |
| E15 | RUNNING | 2008-01-1  13:08:59 | 5NULL | NULL | 5395 | 61 | INSERT INTO t2 VALUES … |
| 51D | RUNNING | 2008-01-1  13:08:47 | 5NULL | NULL | 9807 | 8 | INSERT INTO t2 VALUES … |

The following table shows the contents of the data\_lock\_waits table for a system running a heavy workload.

|  |  |  |  |
| --- | --- | --- | --- |
| **requesting** **trx** **id** | **requested** **lock** **id** | **blocking** **trx** **id** | **blocking** **lock** **id** |
| 77F | 77F:806 | 77E | 77E:806 |
| 77F | 77F:806 | 77D | 77D:806 |
| 77F | 77F:806 | 77B | 77B:806 |



|  |  |  |  |
| --- | --- | --- | --- |
| **requesting** **trx** **id** | **requested** **lock** **id** | **blocking** **trx** **id** | **blocking** **lock** **id** |
| 77E | 77E:806 | 77D | 77D:806 |
| 77E | 77E:806 | 77B | 77B:806 |
| 77D | 77D:806 | 77B | 77B:806 |
| 77B | 77B:733:12:1 | 77A | 77A:733:12:1 |
| E56 | E56:743:6:2 | E55 | E55:743:6:2 |
| E55 | E55:743:38:2 | 19C | 19C:743:38:2 |

The following table shows the contents of the data\_locks table for a system running a heavy

workload.

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| **lock** **id** | **lock** **trx** **id** | **lock** **mode** | **lock** **type** | **lock**  **schema** | **lock** **table** | **lock** **index** | **lock** **data** |
| 77F:806 | 77F | AUTO\_INC | TABLE | test | t09 | NULL | NULL |
| 77E:806 | 77E | AUTO\_INC | TABLE | test | t09 | NULL | NULL |
| 77D:806 | 77D | AUTO\_INC | TABLE | test | t09 | NULL | NULL |
| 77B:806 | 77B | AUTO\_INC | TABLE | test | t09 | NULL | NULL |
| 77B:733:1 | 27:71B | X | RECORD | test | t09 | PRIMARY | supremum  pseudo-  record |
| 77A:733:1 | 27:71A | X | RECORD | test | t09 | PRIMARY | supremum  pseudo-  record |
| E56:743:6 | :E256 | S | RECORD | test | t2 | PRIMARY | 0, 0 |
| E55:743:6 | :E255 | X | RECORD | test | t2 | PRIMARY | 0, 0 |
| E55:743:3 | 8E:525 | S | RECORD | test | t2 | PRIMARY | 1922,  1922 |
| 19C:743:3 | 81:92C | X | RECORD | test | t2 | PRIMARY | 1922,  1922 |

**15.15.2.2** **InnoDB** **Lock** **and** **Lock-Wait** **Information**

**Note**

This section describes locking information as exposed by the Performance Schema data\_locks and data\_lock\_waits tables, which supersede the INFORMATION\_SCHEMA INNODB\_LOCKS and INNODB\_LOCK\_WAITS tables in MySQL 8.0. For similar discussion written in terms of the older INFORMATION\_SCHEMA tables, see [InnoDB Lock and Lock-Wait Information](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-understanding-innodb-locking.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

When a transaction updates a row in a table, or locks it with SELECT FOR UPDATE, InnoDB establishes a list or queue of locks on that row. Similarly, InnoDB maintains a list of locks on a table for table-level locks. If a second transaction wants to update a row or lock a table already locked by a prior transaction in an incompatible mode, InnoDB adds a lock request for the row to the corresponding queue. For a lock to be acquired by a transaction, all incompatible lock requests previously entered into the lock queue for that row or table must be removed (which occurs when the transactions holding or requesting those locks either commit or roll back).

A transaction may have any number of lock requests for different rows or tables. At any given time, a transaction may request a lock that is held by another transaction, in which case it is blocked by that other transaction. The requesting transaction must wait for the transaction that holds the blocking lock to commit or roll back. If a transaction is not waiting for a lock, it is in a RUNNING state. If a transaction



is waiting for a lock, it is in a LOCK WAIT state. (The INFORMATION\_SCHEMA INNODB\_TRX table indicates transaction state values.)

The Performance Schema data\_locks table holds one or more rows for each LOCK WAIT transaction, indicating any lock requests that prevent its progress. This table also contains one row describing each lock in a queue of locks pending for a given row or table. The Performance Schema data\_lock\_waits table shows which locks already held by a transaction are blocking locks requested by other transactions.

**15.15.2.3** **Persistence** **and** **Consistency** **of** **InnoDB** **Transaction** **and** **Locking** **Information**

**Note**

This section describes locking information as exposed by the Performance Schema data\_locks and data\_lock\_waits tables, which supersede the INFORMATION\_SCHEMA INNODB\_LOCKS and INNODB\_LOCK\_WAITS tables in MySQL 8.0. For similar discussion written in terms of the older INFORMATION\_SCHEMA tables, see [Persistence and Consistency of InnoDB](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-internal-data.html) [Transaction and Locking Information](https://dev.mysql.com/doc/refman/5.7/en/innodb-information-schema-internal-data.html), in [MySQL 5.7 Reference Manual](https://dev.mysql.com/doc/refman/5.7/en/).

The data exposed by the transaction and locking tables ( INFORMATION\_SCHEMA INNODB\_TRX table, Performance Schema data\_locks and data\_lock\_waits tables) represents a glimpse into fast- changing data. This is not like user tables, where the data changes only when application-initiated updates occur. The underlying data is internal system-managed data, and can change very quickly:

• Data might not be consistent between the INNODB\_TRX, data\_locks, and data\_lock\_waits tables.

The data\_locks and data\_lock\_waits tables expose live data from the InnoDB storage engine, to provide lock information about the transactions in the INNODB\_TRX table. Data retrieved from the lock tables exists when the SELECT is executed, but might be gone or changed by the time the query result is consumed by the client.

Joining data\_locks with data\_lock\_waits can show rows in data\_lock\_waits that identify a parent row in data\_locks that no longer exists or does not exist yet.

• Data in the transaction and locking tables might not be consistent with data in the INFORMATION\_SCHEMA PROCESSLIST table or Performance Schema threads table.

For example, you should be careful when comparing data in the InnoDB transaction and locking tables with data in the PROCESSLIST table. Even if you issue a single SELECT (joining INNODB\_TRX and PROCESSLIST, for example), the content of those tables is generally not consistent. It is possible for INNODB\_TRX to reference rows that are not present in PROCESSLIST or for the currently executing SQL query of a transaction shown in INNODB\_TRX.TRX\_QUERY to differ from the one in

PROCESSLIST.INFO.

**15.15.3** **InnoDB** **INFORMATION\_SCHEMA** **Schema** **Object** **Tables**

You can extract metadata about schema objects managed by InnoDB using InnoDB INFORMATION\_SCHEMA tables. This information comes from the data dictionary. Traditionally, you would get this type of information using the techniques from [Section 15.17, “InnoDB Monitors”](#_bookmark247) , setting up InnoDB monitors and parsing the output from the SHOW ENGINE INNODB STATUS statement. The InnoDB INFORMATION\_SCHEMA table interface allows you to query this data using SQL.

InnoDB INFORMATION\_SCHEMA schema object tables include the tables listed below.

INNODB\_DATAFILES

INNODB\_TABLESTATS

INNODB\_FOREIGN

INNODB\_COLUMNS

INNODB\_INDEXES

INNODB\_FIELDS

**ENGINE** **=** **InnoDB;**

mysql> **CREATE** **INDEX** **i1** **ON** **t1(col1);**

INNODB\_TABLESPACES

INNODB\_TABLESPACES\_BRIEF

INNODB\_FOREIGN\_COLS

INNODB\_TABLES

The table names are indicative of the type of data provided:

• INNODB\_TABLES provides metadata about InnoDB tables.

• INNODB\_COLUMNS provides metadata about InnoDB table columns.

• INNODB\_INDEXES provides metadata about InnoDB indexes.

• INNODB\_FIELDS provides metadata about the key columns (fields) of InnoDB indexes.

• INNODB\_TABLESTATS provides a view of low-level status information about InnoDB tables that is derived from in-memory data structures.

• INNODB\_DATAFILES provides data file path information for InnoDB file-per-table and general tablespaces.

• INNODB\_TABLESPACES provides metadata about InnoDB file-per-table, general, and undo tablespaces.

• INNODB\_TABLESPACES\_BRIEF provides a subset of metadata about InnoDB tablespaces.

• INNODB\_FOREIGN provides metadata about foreign keys defined on InnoDB tables.

• INNODB\_FOREIGN\_COLS provides metadata about the columns of foreign keys that are defined on InnoDB tables.

InnoDB INFORMATION\_SCHEMA schema object tables can be joined together through fields such as TABLE\_ID, INDEX\_ID, and SPACE, allowing you to easily retrieve all available data for an object you want to study or monitor.

Refer to the InnoDB INFORMATION\_SCHEMA documentation for information about the columns of each table.

**Example** **15.2** **InnoDB** **INFORMATION\_SCHEMA** **Schema** **Object** **Tables**

This example uses a simple table (t1) with a single index (i1) to demonstrate the type of metadata found in the InnoDB INFORMATION\_SCHEMA schema object tables.

1. Create a test database and table t1:

mysql> **CREATE** **DATABASE** **test;**

mysql> **USE** **test;**

mysql> **CREATE** **TABLE** **t1** **(**

**col1** **INT,**

**col2** **CHAR(10),**

**col3** **VARCHAR(10))**

2. After creating the table t1, query INNODB\_TABLES to locate the metadata for test/t1:

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLES** **WHERE** **NAME='test/t1'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 71

NAME: test/t1

FLAG: 1

N\_COLS: 6

SPACE: 57

ROW\_FORMAT: Compact

ZIP\_PAGE\_SIZE: 0

INSTANT\_COLS: 0

Table t1 has a TABLE\_ID of 71. The FLAG field provides bit level information about table format and storage characteristics. There are six columns, three of which are hidden columns created by InnoDB (DB\_ROW\_ID, DB\_TRX\_ID, and DB\_ROLL\_PTR). The ID of the table's SPACE is 57 (a value of 0 would indicate that the table resides in the system tablespace). The ROW\_FORMAT is Compact. ZIP\_PAGE\_SIZE only applies to tables with a Compressed row format. INSTANT\_COLS shows number of columns in the table prior to adding the first instant column using ALTER TABLE ... ADD COLUMN with ALGORITHM=INSTANT.

3. Using the TABLE\_ID information from INNODB\_TABLES, query the INNODB\_COLUMNS table for information about the table's columns.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_COLUMNS** **where** **TABLE\_ID** **=** **71\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 71

NAME: col1

POS: 0

MTYPE: 6

PRTYPE: 1027

LEN: 4

HAS\_DEFAULT: 0

DEFAULT\_VALUE: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 71

NAME: col2

POS: 1

MTYPE: 2

PRTYPE: 524542

LEN: 10

HAS\_DEFAULT: 0

DEFAULT\_VALUE: NULL

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 71

NAME: col3

POS: 2

MTYPE: 1

PRTYPE: 524303

LEN: 10

HAS\_DEFAULT: 0

DEFAULT\_VALUE: NULL

In addition to the TABLE\_ID and column NAME, INNODB\_COLUMNS provides the ordinal position (POS) of each column (starting from 0 and incrementing sequentially), the column MTYPE or “main type” (6 = INT, 2 = CHAR, 1 = VARCHAR), the PRTYPE or “precise type” (a binary value with bits that represent the MySQL data type, character set code, and nullability), and the column length (LEN). The HAS\_DEFAULT and DEFAULT\_VALUE columns only apply to columns added instantly using ALTER TABLE ... ADD COLUMN with ALGORITHM=INSTANT.

4. Using the TABLE\_ID information from INNODB\_TABLES once again, query INNODB\_INDEXES for information about the indexes associated with table t1.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_INDEXES** **WHERE** **TABLE\_ID** **=** **71** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INDEX\_ID: 111

NAME: GEN\_CLUST\_INDEX

TABLE\_ID: 71

TYPE: 1

N\_FIELDS: 0

PAGE\_NO: 3

SPACE: 57

MERGE\_THRESHOLD: 50

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INDEX\_ID: 112

NAME: i1

TABLE\_ID: 71

TYPE: 0

N\_FIELDS: 1

PAGE\_NO: 4

SPACE: 57

MERGE\_THRESHOLD: 50

INNODB\_INDEXES returns data for two indexes. The first index is GEN\_CLUST\_INDEX, which is a clustered index created by InnoDB if the table does not have a user-defined clustered index. The second index (i1) is the user-defined secondary index.

The INDEX\_ID is an identifier for the index that is unique across all databases in an instance. The TABLE\_ID identifies the table that the index is associated with. The index TYPE value indicates the type of index (1 = Clustered Index, 0 = Secondary index). The N\_FILEDS value is the number of fields that comprise the index. PAGE\_NO is the root page number of the index B-tree, and SPACE is the ID of the tablespace where the index resides. A nonzero value indicates that the index does not reside in the system tablespace. MERGE\_THRESHOLD defines a percentage threshold value for the amount of data in an index page. If the amount of data in an index page falls below the this value (the default is 50%) when a row is deleted or when a row is shortened by an update operation, InnoDB attempts to merge the index page with a neighboring index page.

5. Using the INDEX\_ID information from INNODB\_INDEXES, query INNODB\_FIELDS for information about the fields of index i1.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FIELDS** **where** **INDEX\_ID** **=** **112** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

INDEX\_ID: 112

NAME: col1

POS: 0

INNODB\_FIELDS provides the NAME of the indexed field and its ordinal position within the index. If the index (i1) had been defined on multiple fields, INNODB\_FIELDS would provide metadata for each of the indexed fields.

6. Using the SPACE information from INNODB\_TABLES, query INNODB\_TABLESPACES table for information about the table's tablespace.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLESPACES** **WHERE** **SPACE** **=** **57** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: 57

NAME: test/t1

FLAG: 16417

ROW\_FORMAT: Dynamic

PAGE\_SIZE: 16384

ZIP\_PAGE\_SIZE: 0

SPACE\_TYPE: Single

FS\_BLOCK\_SIZE: 4096

FILE\_SIZE: 114688

ALLOCATED\_SIZE: 98304

AUTOEXTEND\_SIZE: 0

SERVER\_VERSION: 8.0.23

SPACE\_VERSION: 1

ENCRYPTION: N

STATE: normal

In addition to the SPACE ID of the tablespace and the NAME of the associated table, INNODB\_TABLESPACES provides tablespace FLAG data, which is bit level information about tablespace format and storage characteristics. Also provided are tablespace ROW\_FORMAT, PAGE\_SIZE, and several other tablespace metadata items.

7. Using the SPACE information from INNODB\_TABLES once again, query INNODB\_DATAFILES for the location of the tablespace data file.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_DATAFILES** **WHERE** **SPACE** **=** **57** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

SPACE: [57](#_bookmark248)

PATH: ./test/t1.ibd

The datafile is located in the test directory under MySQL's data directory. If a file-per- table tablespace were created in a location outside the MySQL data directory using the DATA

**ON** **DELETE** **CASCADE)** **ENGINE=INNODB;**

DIRECTORY clause of the CREATE TABLE statement, the tablespace PATH would be a fully qualified directory path.

8. As a final step, insert a row into table t1 (TABLE\_ID = 71) and view the data in the INNODB\_TABLESTATS table. The data in this table is used by the MySQL optimizer to calculate which index to use when querying an InnoDB table. This information is derived from in-memory data structures.

mysql> **INSERT** **INTO** **t1** **VALUES(5,** **'abc',** **'def');**

Query OK, 1 row affected (0.06 sec)

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TABLESTATS** **where** **TABLE\_ID** **=** **71** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 71

NAME: test/t1

STATS\_INITIALIZED: Initialized

NUM\_ROWS: 1

CLUST\_INDEX\_SIZE: 1

OTHER\_INDEX\_SIZE: 0

MODIFIED\_COUNTER: 1

AUTOINC: 0

REF\_COUNT: 1

The STATS\_INITIALIZED field indicates whether or not statistics have been collected for the table. NUM\_ROWS is the current estimated number of rows in the table. The CLUST\_INDEX\_SIZE and OTHER\_INDEX\_SIZE fields report the number of pages on disk that store clustered and secondary indexes for the table, respectively. The MODIFIED\_COUNTER value shows the number of rows modified by DML operations and cascade operations from foreign keys. The AUTOINC value is the next number to be issued for any autoincrement-based operation. There are no autoincrement columns defined on table t1, so the value is 0. The REF\_COUNT value is a counter. When the counter reaches 0, it signifies that the table metadata can be evicted from the table cache.

**Example** **15.3** **Foreign** **Key** **INFORMATION\_SCHEMA** **Schema** **Object** **Tables**

The INNODB\_FOREIGN and INNODB\_FOREIGN\_COLS tables provide data about foreign key relationships. This example uses a parent table and child table with a foreign key relationship to demonstrate the data found in the INNODB\_FOREIGN and INNODB\_FOREIGN\_COLS tables.

1. Create the test database with parent and child tables:

mysql> **CREATE** **DATABASE** **test;**

mysql> **USE** **test;**

mysql> **CREATE** **TABLE** **parent** **(id** **INT** **NOT** **NULL,**

**PRIMARY** **KEY** **(id))** **ENGINE=INNODB;**

mysql> **CREATE** **TABLE** **child** **(id** **INT,** **parent\_id** **INT,**

**INDEX** **par\_ind** **(parent\_id),**

**CONSTRAINT** **fk1**

**FOREIGN** **KEY** **(parent\_id)** **REFERENCES** **parent(id)**

2. After the parent and child tables are created, query INNODB\_FOREIGN and locate the foreign key data for the test/child and test/parent foreign key relationship:

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FOREIGN** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID: test/fk1

FOR\_NAME: test/child

REF\_NAME: test/parent

N\_COLS: 1

TYPE: 1

Metadata includes the foreign key ID (fk1), which is named for the CONSTRAINT that was defined on the child table. The FOR\_NAME is the name of the child table where the foreign key is defined. REF\_NAME is the name of the parent table (the “referenced” table). N\_COLS is the

**ORDER** **BY** **a.NAME** **DESC;**

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

| NAME | ROW\_FORMAT | page\_size | pk\_mb | secidx\_mb |

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

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

employees/titles

employees/salaries

employees/employees

employees/dept\_manager

employees/dept\_emp

employees/departments

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

Dynamic

Dynamic

Dynamic

Dynamic

Dynamic

Dynamic

16384

16384

16384

16384

16384

16384

20

93

15

0

12

0

11

34

0

0

10

0

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

number of columns in the foreign key index. TYPE is a numerical value representing bit flags that provide additional information about the foreign key column. In this case, the TYPE value is 1, which indicates that the ON DELETE CASCADE option was specified for the foreign key. See the INNODB\_FOREIGN table definition for more information about TYPE values.

3. Using the foreign key ID, query INNODB\_FOREIGN\_COLS to view data about the columns of the foreign key.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FOREIGN\_COLS** **WHERE** **ID** **=** **'test/fk1'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

ID: test/fk1

FOR\_COL\_NAME: parent\_id

REF\_COL\_NAME: id

POS: 0

FOR\_COL\_NAME is the name of the foreign key column in the child table, and REF\_COL\_NAME is the name of the referenced column in the parent table. The POS value is the ordinal position of the key field within the foreign key index, starting at zero.

**Example** **15.4** **Joining** **InnoDB** **INFORMATION\_SCHEMA** **Schema** **Object** **Tables**

This example demonstrates joining three InnoDB INFORMATION\_SCHEMA schema object tables ( INNODB\_TABLES, INNODB\_TABLESPACES, and INNODB\_TABLESTATS) to gather file format, row format, page size, and index size information about tables in the employees sample database.

The following table aliases are used to shorten the query string:

• INFORMATION\_SCHEMA.INNODB\_TABLES: a

• INFORMATION\_SCHEMA.INNODB\_TABLESPACES: b

• INFORMATION\_SCHEMA.INNODB\_TABLESTATS: c

An IF() control flow function is used to account for compressed tables. If a table is compressed, the index size is calculated using ZIP\_PAGE\_SIZE rather than PAGE\_SIZE. CLUST\_INDEX\_SIZE and OTHER\_INDEX\_SIZE, which are reported in bytes, are divided by 1024\*1024 to provide index sizes in megabytes (MBs). MB values are rounded to zero decimal spaces using the ROUND() function.

mysql> **SELECT** **a** **.NAME,** **a** **.ROW\_FORMAT,**

**@page\_size** **:=**

**IF(a.ROW\_FORMAT='Compressed',**

**b.ZIP\_PAGE\_SIZE,** **b.PAGE\_SIZE)**

**AS** **page\_size,**

**ROUND((@page\_size** **\*** **c.CLUST\_INDEX\_SIZE)**

**/(1024\*1024))** **AS** **pk\_mb,**

**ROUND((@page\_size** **\*** **c.OTHER\_INDEX\_SIZE)**

**/(1024\*1024))** **AS** **secidx\_mb**

**FROM** **INFORMATION\_SCHEMA.INNODB\_TABLES** **a**

**INNER** **JOIN** **INFORMATION\_SCHEMA.INNODB\_TABLESPACES** **b** **on** **a.NAME** **=** **b.NAME**

**INNER** **JOIN** **INFORMATION\_SCHEMA** **.INNODB\_TABLESTATS** **c** **on** **b** **.NAME** **=** **c** **.NAME**

**WHERE** **a** **.NAME** **LIKE** **'employees/%'**

**15.15.4** **InnoDB** **INFORMATION\_SCHEMA** **FULLTEXT** **Index** **Tables**

The following tables provide metadata for FULLTEXT indexes:

mysql> **SHOW** **TABLES** **FROM** **INFORMATION\_SCHEMA** **LIKE** **'INNODB\_FT%';**



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

| Tables\_in\_INFORMATION\_SCHEMA (INNODB\_FT%) |

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

|  |  |  |
| --- | --- | --- |
| | | INNODB\_FT\_CONFIG | | |
| | | INNODB\_FT\_BEING\_DELETED | | |
| | | INNODB\_FT\_DELETED | | |
| | | INNODB\_FT\_DEFAULT\_STOPWORD | | |
| | | INNODB\_FT\_INDEX\_TABLE | | |
| | | INNODB\_FT\_INDEX\_CACHE | | |

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

**Table** **Overview**

• INNODB\_FT\_CONFIG: Provides metadata about the FULLTEXT index and associated processing for an InnoDB table.

• INNODB\_FT\_BEING\_DELETED: Provides a snapshot of the INNODB\_FT\_DELETED table; it is used only during an OPTIMIZE TABLE maintenance operation. When OPTIMIZE TABLE is run, the INNODB\_FT\_BEING\_DELETED table is emptied, and DOC\_ID values are removed from the INNODB\_FT\_DELETED table. Because the contents of INNODB\_FT\_BEING\_DELETED typically

have a short lifetime, this table has limited utility for monitoring or debugging. For information about running OPTIMIZE TABLE on tables with FULLTEXT indexes, see Section 12. 10.6, “Fine-Tuning MySQL Full-Text Search” .

• INNODB\_FT\_DELETED: Stores rows that are deleted from the FULLTEXT index for an InnoDB table. To avoid expensive index reorganization during DML operations for an InnoDB FULLTEXT index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an OPTIMIZE TABLE statement for the InnoDB table.

• INNODB\_FT\_DEFAULT\_STOPWORD: Holds a list of stopwords that are used by default when creating a FULLTEXT index on InnoDB tables.

For information about the INNODB\_FT\_DEFAULT\_STOPWORD table, see Section 12.10.4, “Full-Text Stopwords” .

• INNODB\_FT\_INDEX\_TABLE: Provides information about the inverted index used to process text searches against the FULLTEXT index of an InnoDB table.

• INNODB\_FT\_INDEX\_CACHE: Provides token information about newly inserted rows in a FULLTEXT index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when OPTIMIZE TABLE is run, when the server is shut down, or when the cache size exceeds a limit defined by the [innodb\_ft\_cache\_size](#_bookmark170) or [innodb\_ft\_total\_cache\_size](#_bookmark171) system variable.

**Note**

With the exception of the INNODB\_FT\_DEFAULT\_STOPWORD table, these tables are empty initially. Before querying any of them, set the value of the [innodb\_ft\_aux\_table](#_bookmark168) system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example, test/ articles).

**Example** **15.5** **InnoDB** **FULLTEXT** **Index** **INFORMATION\_SCHEMA** **Tables**

This example uses a table with a FULLTEXT index to demonstrate the data contained in the FULLTEXT index INFORMATION\_SCHEMA tables.

1. Create a table with a FULLTEXT index and insert some data:

mysql> **CREATE** **TABLE** **articles** **(**

**id** **INT** **UNSIGNED** **AUTO\_INCREMENT** **NOT** **NULL** **PRIMARY** **KEY,**

**title** **VARCHAR(200),**

**body** **TEXT,**

**FULLTEXT** **(title,body)**

**('MySQL** **Security','When** **configured** **properly,** **MySQL** **...');**

**)** **ENGINE=InnoDB;**

mysql> **INSERT** **INTO** **articles** **(title,body)** **VALUES**

**('MySQL** **Tutorial','DBMS** **stands** **for** **DataBase** **.** **.** **.** **'),**

**('How** **To** **Use** **MySQL** **Well','After** **you** **went** **through** **a** **.** **.** **.** **'),**

**('Optimizing** **MySQL','In** **this** **tutorial** **we** **show** **.** **.** **.** **'),**

**('1001** **MySQL** **Tricks','1** **.** **Never** **run** **mysqld** **as** **root** **.** **2.** **.** **.** **.** **'),**

**('MySQL** **vs** **.** **YourSQL','In** **the** **following** **database** **comparison** **.** **.** **.** **'),**

2. Set the [innodb\_ft\_aux\_table](#_bookmark168) variable to the name of the table with the FULLTEXT index. If this variable is not set, the InnoDB FULLTEXT INFORMATION\_SCHEMA tables are empty, with the exception of INNODB\_FT\_DEFAULT\_STOPWORD.

mysql> **SET** **GLOBAL** **innodb\_ft\_aux\_table** **=** **'test/articles';**

3. Query the INNODB\_FT\_INDEX\_CACHE table, which shows information about newly inserted rows in a FULLTEXT index. To avoid expensive index reorganization during DML operations, data for newly inserted rows remains in the FULLTEXT index cache until OPTIMIZE TABLE is run (or until the server is shut down or cache limits are exceeded).

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_CACHE** **LIMIT** **5;**

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

| WORD | FIRST\_DOC\_ID | LAST\_DOC\_ID | DOC\_COUNT | DOC\_ID | POSITION |

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

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

0

22

44

20

31

1001

after

comparison

configured

database

1

1

1

1

2

5

3

6

7

2

5

3

6

7

6

5

3

6

7

2

|

|

|

|

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

4. Enable the [innodb\_optimize\_fulltext\_only](#_bookmark212) system variable and run OPTIMIZE TABLE on the table that contains the FULLTEXT index. This operation flushes the contents of the FULLTEXT index cache to the main FULLTEXT index. [innodb\_optimize\_fulltext\_only](#_bookmark212) changes the way the OPTIMIZE TABLE statement operates on InnoDB tables, and is intended to be enabled temporarily, during maintenance operations on InnoDB tables with FULLTEXT indexes.

mysql> **SET** **GLOBAL** **innodb\_optimize\_fulltext\_only=ON;**

mysql> **OPTIMIZE** **TABLE** **articles;**

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

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

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

| test .articles | optimize | status | OK |

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

5. Query the INNODB\_FT\_INDEX\_TABLE table to view information about data in the main FULLTEXT index, including information about the data that was just flushed from the FULLTEXT index cache.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_TABLE** **LIMIT** **5;**

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

| WORD | FIRST\_DOC\_ID | LAST\_DOC\_ID | DOC\_COUNT | DOC\_ID | POSITION |

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

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

1001

after

comparison

configured

database

0

22

44

20

31

1

1

1

1

2

5

3

6

7

6

5

3

6

7

2

5

3

6

7

2

|

|

|

|

|

|

|

|

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

The INNODB\_FT\_INDEX\_CACHE table is now empty since the OPTIMIZE TABLE operation flushed the FULLTEXT index cache.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_INDEX\_CACHE** **LIMIT** **5;**

Empty set (0.00 sec)

6. Delete some records from the test/articles table.

mysql> **DELETE** **FROM** **test.articles** **WHERE** **id** **<** **4;**

7. Query the INNODB\_FT\_DELETED table. This table records rows that are deleted from the FULLTEXT index. To avoid expensive index reorganization during DML operations, information about newly deleted records is stored separately, filtered out of search results when you do a text search, and removed from the main search index when you run OPTIMIZE TABLE.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_DELETED;**

+--------+

| DOC\_ID |

+--------+

|

|

|

|

2

3

4

|

|

+--------+

8. Run OPTIMIZE TABLE to remove the deleted records.

mysql> **OPTIMIZE** **TABLE** **articles;**

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

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

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

| test .articles | optimize | status | OK |

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

The INNODB\_FT\_DELETED table should now be empty.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_DELETED;**

Empty set (0.00 sec)

9. Query the INNODB\_FT\_CONFIG table. This table contains metadata about the FULLTEXT index and related processing:

• optimize\_checkpoint\_limit: The number of seconds after which an OPTIMIZE TABLE run stops.

• synced\_doc\_id: The next DOC\_ID to be issued.

• stopword\_table\_name: The *database/table* name for a user-defined stopword table. The VALUE column is empty if there is no user-defined stopword table.

• use\_stopword: Indicates whether a stopword table is used, which is defined when the FULLTEXT index is created.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_FT\_CONFIG;**

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

| KEY | VALUE |

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

|

|

|

|

|

optimize\_checkpoint\_limit

synced\_doc\_id

stopword\_table\_name

use\_stopword

|

|

|

|

180

8

|

|

|

1

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

10. Disable [innodb\_optimize\_fulltext\_only](#_bookmark212), since it is intended to be enabled only temporarily: mysql> **SET** **GLOBAL** **innodb\_optimize\_fulltext\_only=OFF;**

**15.15.5** **InnoDB** **INFORMATION\_SCHEMA** **Buffer** **Pool** **Tables**

The InnoDB INFORMATION\_SCHEMA buffer pool tables provide buffer pool status information and metadata about the pages within the InnoDB buffer pool.

The InnoDB INFORMATION\_SCHEMA buffer pool tables include those listed below:

mysql> **SHOW** **TABLES** **FROM** **INFORMATION\_SCHEMA** **LIKE** **'INNODB\_BUFFER%';**

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



| Tables\_in\_INFORMATION\_SCHEMA (INNODB\_BUFFER%) |

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

|  |  |  |
| --- | --- | --- |
| |  |  | | INNODB\_BUFFER\_PAGE\_LRU  INNODB\_BUFFER\_PAGE  INNODB\_BUFFER\_POOL\_STATS | |  |  | |

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

**Table** **Overview**

• INNODB\_BUFFER\_PAGE: Holds information about each page in the InnoDB buffer pool.

• INNODB\_BUFFER\_PAGE\_LRU: Holds information about the pages in the InnoDB buffer pool, in particular how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full. The INNODB\_BUFFER\_PAGE\_LRU table has the same columns as the INNODB\_BUFFER\_PAGE table, except that the INNODB\_BUFFER\_PAGE\_LRU table has an LRU\_POSITION column instead of a BLOCK\_ID column.

• INNODB\_BUFFER\_POOL\_STATS: Provides buffer pool status information. Much of the same information is provided by SHOW ENGINE INNODB STATUS output, or may be obtained using InnoDB buffer pool server status variables.

**Warning**

Querying the INNODB\_BUFFER\_PAGE or INNODB\_BUFFER\_PAGE\_LRU table can affect performance. Do not query these tables on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

**Example** **15.6** **Querying** **System** **Data** **in** **the** **INNODB\_BUFFER\_PAGE** **Table**

This query provides an approximate count of pages that contain system data by excluding pages where the TABLE\_NAME value is either NULL or includes a slash / or period . in the table name, which indicates a user-defined table.

mysql> **SELECT** **COUNT(\*)** **FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE**

**WHERE** **TABLE\_NAME** **IS** **NULL** **OR** **(INSTR(TABLE\_NAME,** **'/')** **=** **0** **AND** **INSTR(TABLE\_NAME,** **'** **.** **')** **=** **0);**

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

| COUNT(\*) |

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

|  |  |
| --- | --- |
| | | 1516 | |

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

This query returns the approximate number of pages that contain system data, the total number of buffer pool pages, and an approximate percentage of pages that contain system data.

mysql> **SELECT**

**(SELECT** **COUNT(\*)** **FROM** **INFORMATION\_SCHEMA** **.INNODB\_BUFFER\_PAGE**

**WHERE** **TABLE\_NAME** **IS** **NULL** **OR** **(INSTR(TABLE\_NAME,** **'/')** **=** **0** **AND** **INSTR(TABLE\_NAME,** **'** **.** **')** **=** **0)**

**)** **AS** **system\_pages,**

**(**

**SELECT** **COUNT(\*)**

**FROM** **INFORMATION\_SCHEMA** **.INNODB\_BUFFER\_PAGE**

**)** **AS** **total\_pages,**

**(**

**SELECT** **ROUND((system\_pages/total\_pages)** **\*** **100)**

**)** **AS** **system\_page\_percentage;**

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

| system\_pages | total\_pages | system\_page\_percentage |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | | 295 | | | 8192 | | | 4 | |

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

The type of system data in the buffer pool can be determined by querying the PAGE\_TYPE value. For example, the following query returns eight distinct PAGE\_TYPE values among the pages that contain system data:

**WHERE** **TABLE\_NAME** **IS** **NOT** **NULL** **AND** **TABLE\_NAME** **NOT** **LIKE** **'%INNODB\_TABLES%';**

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

| COUNT(\*) |

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

| 7897 |

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

**)** **AS** **user\_page\_percentage;**

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

| user\_pages | total\_pages | user\_page\_percentage |

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

| 7897 | 8192 | 96 |

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

mysql> **SELECT** **DISTINCT** **PAGE\_TYPE** **FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE**

**WHERE** **TABLE\_NAME** **IS** **NULL** **OR** **(INSTR(TABLE\_NAME,** **'/')** **=** **0** **AND** **INSTR(TABLE\_NAME,** **'** **.** **')** **=** **0);**

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

|

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

SYSTEM

IBUF\_BITMAP

UNKNOWN

FILE\_SPACE\_HEADER

INODE

UNDO\_LOG

ALLOCATED

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

**Example** **15.7** **Querying** **User** **Data** **in** **the** **INNODB\_BUFFER\_PAGE** **Table**

| PAGE\_TYPE

|

|

|

|

|

|

|

|

|

|

|

|

|

|

This query provides an approximate count of pages containing user data by counting pages where the TABLE\_NAME value is NOT NULL and NOT LIKE '%INNODB\_TABLES%'.

mysql> **SELECT** **COUNT(\*)** **FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE**

This query returns the approximate number of pages that contain user data, the total number of buffer pool pages, and an approximate percentage of pages that contain user data.

mysql> **SELECT**

**FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE**

IS NOT NULL AND (INSTR(TABLE\_NAME, '/') > 0 OR INSTR(TABLE\_NAME, '. ') > 0)

**SELECT** **ROUND((user\_pages/total\_pages)** **\*** **100)**

**FROM** **information\_schema.INNODB\_BUFFER\_PAGE**

**)** **AS** **user\_pages,**

**(**

**SELECT** **COUNT(\*)**

**(SELECT** **COUNT(\*)**

WHERE TABLE\_NAME

**)** **AS** **total\_pages,**

**(**

This query identifies user-defined tables with pages in the buffer pool:

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

**AND** **TABLE\_NAME** **NOT** **LIKE** **'`mysql`.`innodb\_%';**

mysql> **SELECT** **DISTINCT** **TABLE\_NAME** **FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE**

**WHERE** **TABLE\_NAME** **IS** **NOT** **NULL** **AND** **(INSTR(TABLE\_NAME,** **'/')** **>** **0** **OR** **INSTR(TABLE\_NAME,** **'** **.** **')** **>** **0)**

| TABLE\_NAME |

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

| `employees` . `salaries` |

| `employees` . `employees` |

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

**Example** **15.8** **Querying** **Index** **Data** **in** **the** **INNODB\_BUFFER\_PAGE** **Table**

For information about index pages, query the INDEX\_NAME column using the name of the index. For example, the following query returns the number of pages and total data size of pages for the emp\_no index that is defined on the employees.salaries table:

mysql> **SELECT** **INDEX\_NAME,** **COUNT(\*)** **AS** **Pages,**

**ROUND(SUM(IF(COMPRESSED\_SIZE** **=** **0,** **@@GLOBAL.innodb\_page\_size,** **COMPRESSED\_SIZE))/1024/1024)**

**AS** **'Total** **Data** **(MB)'**

**FROM** **INFORMATION\_SCHEMA** **.INNODB\_BUFFER\_PAGE**

**WHERE** **INDEX\_NAME='emp\_no'** **AND** **TABLE\_NAME** **=** **'`employees`** **.** **`salaries`';**

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

| INDEX\_NAME | Pages | Total Data (MB) |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| | emp\_no | | | 1609 | | | 25 | |

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

This query returns the number of pages and total data size of pages for all indexes defined on the employees.salaries table:

mysql> **SELECT** **INDEX\_NAME,** **COUNT(\*)** **AS** **Pages,**

**ROUND(SUM(IF(COMPRESSED\_SIZE** **=** **0,** **@@GLOBAL.innodb\_page\_size,** **COMPRESSED\_SIZE))/1024/1024)**

**AS** **'Total** **Data** **(MB)'**

**FROM** **INFORMATION\_SCHEMA** **.INNODB\_BUFFER\_PAGE**

**WHERE** **TABLE\_NAME** **=** **'`employees`** **.`salaries`'**

**GROUP** **BY** **INDEX\_NAME;**

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

| INDEX\_NAME | Pages | Total Data (MB) |

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

|

|

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

emp\_no

PRIMARY

1608

6086

25

95

|

|

|

|

|

|

**Example** **15.9** **Querying** **LRU\_POSITION** **Data** **in** **the** **INNODB\_BUFFER\_PAGE\_LRU** **Table**

The INNODB\_BUFFER\_PAGE\_LRU table holds information about the pages in the InnoDB buffer pool, in particular how they are ordered that determines which pages to evict from the buffer pool when it becomes full. The definition for this page is the same as for INNODB\_BUFFER\_PAGE, except this table has an LRU\_POSITION column instead of a BLOCK\_ID column.

This query counts the number of positions at a specific location in the LRU list occupied by pages of the employees.employees table.

mysql> **SELECT** **COUNT(LRU\_POSITION)** **FROM** **INFORMATION\_SCHEMA.INNODB\_BUFFER\_PAGE\_LRU**

**WHERE** **TABLE\_NAME='`employees`** **.** **`employees`'** **AND** **LRU\_POSITION** **<** **3072;**

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

| COUNT(LRU\_POSITION) |

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

|  |  |
| --- | --- |
| | | 548 | |

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

**Example** **15.10** **Querying** **the** **INNODB\_BUFFER\_POOL\_STATS** **Table**

The INNODB\_BUFFER\_POOL\_STATS table provides information similar to SHOW ENGINE INNODB STATUS and InnoDB buffer pool status variables.

mysql> **SELECT** **\*** **FROM** **information\_schema** **.INNODB\_BUFFER\_POOL\_STATS** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

POOL\_ID: 0

POOL\_SIZE: 8192

FREE\_BUFFERS: 1

DATABASE\_PAGES: 8173

OLD\_DATABASE\_PAGES: 3014

MODIFIED\_DATABASE\_PAGES: 0

PENDING\_DECOMPRESS: 0

PENDING\_READS: 0

PENDING\_FLUSH\_LRU: 0

PENDING\_FLUSH\_LIST: 0

PAGES\_MADE\_YOUNG: 15907

PAGES\_NOT\_MADE\_YOUNG: 3803101

PAGES\_MADE\_YOUNG\_RATE: 0

PAGES\_MADE\_NOT\_YOUNG\_RATE: 0

NUMBER\_PAGES\_READ: 3270

NUMBER\_PAGES\_CREATED: 13176

NUMBER\_PAGES\_WRITTEN: 15109

PAGES\_READ\_RATE: 0

PAGES\_CREATE\_RATE: 0

PAGES\_WRITTEN\_RATE: 0

NUMBER\_PAGES\_GET: 33069332

HIT\_RATE: 0

YOUNG\_MAKE\_PER\_THOUSAND\_GETS: 0

NOT\_YOUNG\_MAKE\_PER\_THOUSAND\_GETS: 0

NUMBER\_PAGES\_READ\_AHEAD: 2713

NUMBER\_READ\_AHEAD\_EVICTED: 0

READ\_AHEAD\_RATE: 0

READ\_AHEAD\_EVICTED\_RATE: 0

LRU\_IO\_TOTAL: 0

LRU\_IO\_CURRENT: 0

UNCOMPRESS\_TOTAL: 0

UNCOMPRESS\_CURRENT: 0

For comparison, SHOW ENGINE INNODB STATUS output and InnoDB buffer pool status variable output is shown below, based on the same data set.

For more information about SHOW ENGINE INNODB STATUS output, see [Section 15.17.3, “InnoDB](#_bookmark249) [Standard Monitor and Lock Monitor Output”](#_bookmark249) .

mysql> **SHOW** **ENGINE** **INNODB** **STATUS** **\G**

...

----------------------

BUFFER POOL AND MEMORY

----------------------

Total large memory allocated 137428992

Dictionary memory allocated 579084

Buffer pool size 8192

Free buffers 1

Database pages 8173

Old database pages 3014

Modified db pages 0

Pending reads 0

Pending writes: LRU 0, flush list 0, single page 0

Pages made young 15907, not young 3803101

0.00 youngs/s, 0.00 non-youngs/s

Pages read 3270, created 13176, written 15109

0.00 reads/s, 0.00 creates/s, 0.00 writes/s

No buffer pool page gets since the last printout

Pages read ahead 0 .00/s, evicted without access 0 .00/s, Random read ahead 0 .00/s

LRU len: 8173, unzip\_LRU len: 0

I/O sum[0]:cur[0], unzip sum[0]:cur[0]

...

For status variable descriptions, see Section 5.1.10, “Server Status Variables” .

mysql> **SHOW** **STATUS** **LIKE** **'Innodb\_buffer%';**

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

|

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

not started

not started

not started

8173

133906432

0

0

15109

1

18

8192

0

2713

0

33069332

558

0

11985961

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

Innodb\_buffer\_pool\_pages\_dirty

Innodb\_buffer\_pool\_bytes\_dirty

Innodb\_buffer\_pool\_pages\_flushed

Innodb\_buffer\_pool\_pages\_free

Innodb\_buffer\_pool\_pages\_misc

Innodb\_buffer\_pool\_pages\_total

Innodb\_buffer\_pool\_read\_ahead\_rnd

Innodb\_buffer\_pool\_read\_ahead

Innodb\_buffer\_pool\_read\_ahead\_evicted

Innodb\_buffer\_pool\_read\_requests

Innodb\_buffer\_pool\_reads

Innodb\_buffer\_pool\_wait\_free

Innodb\_buffer\_pool\_write\_requests

| Innodb\_buffer\_pool\_dump\_status

| Innodb\_buffer\_pool\_load\_status

| Innodb\_buffer\_pool\_resize\_status

| Innodb\_buffer\_pool\_pages\_data

| Innodb\_buffer\_pool\_bytes\_data

| Variable\_name

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

Value

|

**15.15.6** **InnoDB** **INFORMATION\_SCHEMA** **Metrics** **Table**

The INNODB\_METRICS table provides information about InnoDB performance and resource-related counters.

INNODB\_METRICS table columns are shown below. For column descriptions, see Section 26.4.21, “The INFORMATION\_SCHEMA INNODB\_METRICS Table” .



mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 46273

MAX\_COUNT: 46273

MIN\_COUNT: NULL

AVG\_COUNT: 492.2659574468085

COUNT\_RESET: 46273

MAX\_COUNT\_RESET: 46273

MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: NULL

TIME\_ENABLED: 2014-11-28 16:07:53

TIME\_DISABLED: NULL

TIME\_ELAPSED: 94

TIME\_RESET: NULL

STATUS: enabled

TYPE: status\_counter

COMMENT: Number of rows inserted

**Enabling,** **Disabling,** **and** **Resetting** **Counters**

You can enable, disable, and reset counters using the following variables:

• innodb\_monitor\_enable: Enables counters.

SET GLOBAL innodb\_monitor\_enable = [counter-name |module\_name |pattern |all];

• innodb\_monitor\_disable: Disables counters.

SET GLOBAL innodb\_monitor\_disable = [counter-name |module\_name |pattern |all];

• innodb\_monitor\_reset: Resets counter values to zero.

SET GLOBAL innodb\_monitor\_reset = [counter-name |module\_name |pattern |all];

• innodb\_monitor\_reset\_all: Resets all counter values. A counter must be disabled before using innodb\_monitor\_reset\_all.

SET GLOBAL innodb\_monitor\_reset\_all = [counter-name |module\_name |pattern |all];

Counters and counter modules can also be enabled at startup using the MySQL server configuration file. For example, to enable the log module, metadata\_table\_handles\_opened and metadata\_table\_handles\_closed counters, enter the following line in the [mysqld] section of the MySQL server configuration file.

[mysqld]

innodb\_monitor\_enable = log,metadata\_table\_handles\_opened,metadata\_table\_handles\_closed

When enabling multiple counters or modules in a configuration file, specify the innodb\_monitor\_enable variable followed by counter and module names separated by a comma, as shown above. Only the innodb\_monitor\_enable variable can be used in a configuration file. The innodb\_monitor\_disable and innodb\_monitor\_reset variables are supported on the command line only.

**Note**

Because each counter adds a degree of runtime overhead, use counters conservatively on production servers to diagnose specific issues or monitor specific functionality. A test or development server is recommended for more extensive use of counters.

**Counters**

The list of available counters is subject to change. Query the Information Schema INNODB\_METRICS table for counters available in your MySQL server version.

The counters enabled by default correspond to those shown in SHOW ENGINE INNODB STATUS output. Counters shown in SHOW ENGINE INNODB STATUS output are always enabled at a system level but can be disable for the INNODB\_METRICS table. Counter status is not persistent. Unless configured otherwise, counters revert to their default enabled or disabled status when the server is restarted.

If you run programs that would be affected by the addition or removal of counters, it is recommended that you review the releases notes and query the INNODB\_METRICS table to identify those changes as part of your upgrade process.

mysql> **SELECT** **name,** **subsystem,** **status** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **ORDER** **BY** **NAME;**

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

| name

|

subsystem

|

status |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | adaptive\_hash\_pages\_added  adaptive\_hash\_pages\_removed  adaptive\_hash\_rows\_added  | adaptive\_hash\_rows\_deleted\_no\_hash\_entry  | adaptive\_hash\_rows\_removed  | adaptive\_hash\_rows\_updated  adaptive\_hash\_searches  adaptive\_hash\_searches\_btree  | buffer\_data\_reads  | buffer\_data\_written  | buffer\_flush\_adaptive  buffer\_flush\_adaptive\_avg\_pass  buffer\_flush\_adaptive\_avg\_time\_est  | buffer\_flush\_adaptive\_avg\_time\_slot  | buffer\_flush\_adaptive\_avg\_time\_thread  | buffer\_flush\_adaptive\_pages  buffer\_flush\_adaptive\_total\_pages  buffer\_flush\_avg\_page\_rate  | buffer\_flush\_avg\_pass  | buffer\_flush\_avg\_time  | buffer\_flush\_background  buffer\_flush\_background\_pages  buffer\_flush\_background\_total\_pages  | buffer\_flush\_batches  | buffer\_flush\_batch\_num\_scan  | buffer\_flush\_batch\_pages  buffer\_flush\_batch\_scanned  buffer\_flush\_batch\_scanned\_per\_call  | buffer\_flush\_batch\_total\_pages  | buffer\_flush\_lsn\_avg\_rate  | buffer\_flush\_neighbor  buffer\_flush\_neighbor\_pages  buffer\_flush\_neighbor\_total\_pages  | buffer\_flush\_n\_to\_flush\_by\_age  | buffer\_flush\_n\_to\_flush\_by\_dirty\_page  | buffer\_flush\_n\_to\_flush\_requested  buffer\_flush\_pct\_for\_dirty  buffer\_flush\_pct\_for\_lsn  | buffer\_flush\_sync  | buffer\_flush\_sync\_pages  | buffer\_flush\_sync\_total\_pages  buffer\_flush\_sync\_waits  buffer\_LRU\_batches\_evict  | buffer\_LRU\_batches\_flush  | buffer\_LRU\_batch\_evict\_pages  | buffer\_LRU\_batch\_evict\_total\_pages  buffer\_LRU\_batch\_flush\_avg\_pass  buffer\_LRU\_batch\_flush\_avg\_time\_est  | buffer\_LRU\_batch\_flush\_avg\_time\_slot  | buffer\_LRU\_batch\_flush\_avg\_time\_thread  | buffer\_LRU\_batch\_flush\_pages  buffer\_LRU\_batch\_flush\_total\_pages  buffer\_LRU\_batch\_num\_scan  | buffer\_LRU\_batch\_scanned  | buffer\_LRU\_batch\_scanned\_per\_call  | buffer\_LRU\_get\_free\_loops  | buffer LRU get free search | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  adaptive\_hash\_index  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  Buffer | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | disabled  disabled  disabled  disabled  disabled  disabled  enabled  enabled  enabled  enabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|  |  |  |  |
| --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | buffer\_LRU\_get\_free\_waits  buffer\_LRU\_search\_num\_scan  buffer\_LRU\_search\_scanned  buffer\_LRU\_search\_scanned\_per\_call  buffer\_LRU\_single\_flush\_failure\_count  buffer\_LRU\_single\_flush\_num\_scan  buffer\_LRU\_single\_flush\_scanned  buffer\_LRU\_single\_flush\_scanned\_per\_call  buffer\_LRU\_unzip\_search\_num\_scan  buffer\_LRU\_unzip\_search\_scanned  buffer\_LRU\_unzip\_search\_scanned\_per\_call  buffer\_pages\_created  buffer\_pages\_read  buffer\_pages\_written  buffer\_page\_read\_blob  buffer\_page\_read\_fsp\_hdr  buffer\_page\_read\_ibuf\_bitmap  buffer\_page\_read\_ibuf\_free\_list  buffer\_page\_read\_index\_ibuf\_leaf  buffer\_page\_read\_index\_ibuf\_non\_leaf  buffer\_page\_read\_index\_inode  buffer\_page\_read\_index\_leaf  buffer\_page\_read\_index\_non\_leaf  buffer\_page\_read\_other  buffer\_page\_read\_rseg\_array  buffer\_page\_read\_system\_page  buffer\_page\_read\_trx\_system  buffer\_page\_read\_undo\_log  buffer\_page\_read\_xdes  buffer\_page\_read\_zblob  buffer\_page\_read\_zblob2  buffer\_page\_written\_blob  buffer\_page\_written\_fsp\_hdr  buffer\_page\_written\_ibuf\_bitmap  buffer\_page\_written\_ibuf\_free\_list  buffer\_page\_written\_index\_ibuf\_leaf  buffer\_page\_written\_index\_ibuf\_non\_leaf  buffer\_page\_written\_index\_inode  buffer\_page\_written\_index\_leaf  buffer\_page\_written\_index\_non\_leaf  buffer\_page\_written\_on\_log\_no\_waits  buffer\_page\_written\_on\_log\_waits  buffer\_page\_written\_on\_log\_wait\_loops  buffer\_page\_written\_other  buffer\_page\_written\_rseg\_array  buffer\_page\_written\_system\_page  buffer\_page\_written\_trx\_system  buffer\_page\_written\_undo\_log  buffer\_page\_written\_xdes  buffer\_page\_written\_zblob  buffer\_page\_written\_zblob2  buffer\_pool\_bytes\_data  buffer\_pool\_bytes\_dirty  buffer\_pool\_pages\_data  buffer\_pool\_pages\_dirty  buffer\_pool\_pages\_free  buffer\_pool\_pages\_misc  buffer\_pool\_pages\_total  buffer\_pool\_reads  buffer\_pool\_read\_ahead  buffer\_pool\_read\_ahead\_evicted  buffer\_pool\_read\_requests  buffer\_pool\_size  buffer\_pool\_wait\_free  buffer\_pool\_write\_requests  compression\_pad\_decrements  compression\_pad\_increments  compress\_pages\_compressed  compress\_pages\_decompressed  cpu\_n  cpu\_stime\_abs  cpu\_stime\_pct | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | buffer  buffer  buffer  buffer  Buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer\_page\_io  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  buffer  server  buffer  buffer  compression  compression  compression  compression  cpu  cpu  cpu |

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

disabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

disabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

enabled

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

cpu\_utime\_abs

cpu\_utime\_pct

dblwr\_async\_requests

dblwr\_flush\_requests

dblwr\_flush\_wait\_events

dblwr\_sync\_requests

ddl\_background\_drop\_tables

ddl\_log\_file\_alter\_table

ddl\_online\_create\_index

ddl\_pending\_alter\_table

ddl\_sort\_file\_alter\_table

dml\_deletes

dml\_inserts

dml\_reads

dml\_system\_deletes

dml\_system\_inserts

dml\_system\_reads

dml\_system\_updates

dml\_updates

file\_num\_open\_files

ibuf\_merges

ibuf\_merges\_delete

ibuf\_merges\_delete\_mark

ibuf\_merges\_discard\_delete

ibuf\_merges\_discard\_delete\_mark

ibuf\_merges\_discard\_insert

ibuf\_merges\_insert

ibuf\_size

icp\_attempts

icp\_match

icp\_no\_match

icp\_out\_of\_range

index\_page\_discards

index\_page\_merge\_attempts

index\_page\_merge\_successful

index\_page\_reorg\_attempts

index\_page\_reorg\_successful

index\_page\_splits

innodb\_activity\_count

innodb\_background\_drop\_table\_usec

innodb\_dblwr\_pages\_written

innodb\_dblwr\_writes

innodb\_dict\_lru\_count

innodb\_dict\_lru\_usec

innodb\_ibuf\_merge\_usec

innodb\_master\_active\_loops

innodb\_master\_idle\_loops

innodb\_master\_purge\_usec

innodb\_master\_thread\_sleeps

innodb\_mem\_validate\_usec

innodb\_page\_size

innodb\_rwlock\_sx\_os\_waits

innodb\_rwlock\_sx\_spin\_rounds

innodb\_rwlock\_sx\_spin\_waits

innodb\_rwlock\_s\_os\_waits

innodb\_rwlock\_s\_spin\_rounds

innodb\_rwlock\_s\_spin\_waits

innodb\_rwlock\_x\_os\_waits

innodb\_rwlock\_x\_spin\_rounds

innodb\_rwlock\_x\_spin\_waits

lock\_deadlocks

lock\_deadlock\_false\_positives

lock\_deadlock\_rounds

lock\_rec\_grant\_attempts

lock\_rec\_locks

lock\_rec\_lock\_created

lock\_rec\_lock\_removed

lock\_rec\_lock\_requests

lock\_rec\_lock\_waits

lock\_rec\_release\_attempts

lock\_row\_lock\_current\_waits

lock\_row\_lock\_time

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

cpu

cpu

dblwr

dblwr

dblwr

dblwr

ddl

ddl

ddl

ddl

ddl

dml

dml

dml

dml

dml

dml

dml

dml

file\_system

change\_buffer

change\_buffer

change\_buffer

change\_buffer

change\_buffer

change\_buffer

change\_buffer

change\_buffer

icp

icp

icp

icp

index

index

index

index

index

index

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

server

lock

lock

lock

lock

lock

lock

lock

lock

lock

lock

lock

lock

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

enabled

enabled

enabled

enabled

disabled

disabled

disabled

disabled

enabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

disabled

disabled

disabled

disabled

disabled

enabled

disabled

disabled

disabled

enabled

disabled

disabled

disabled

disabled

disabled

disabled

disabled

enabled

enabled

enabled

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

lock\_row\_lock\_time\_avg

lock\_row\_lock\_time\_max

lock\_row\_lock\_waits

lock\_schedule\_refreshes

lock\_table\_locks

lock\_table\_lock\_created

lock\_table\_lock\_removed

lock\_table\_lock\_waits

lock\_threads\_waiting

lock\_timeouts

log\_checkpoints

log\_concurrency\_margin

log\_flusher\_no\_waits

log\_flusher\_waits

log\_flusher\_wait\_loops

log\_flush\_avg\_time

log\_flush\_lsn\_avg\_rate

log\_flush\_max\_time

log\_flush\_notifier\_no\_waits

log\_flush\_notifier\_waits

log\_flush\_notifier\_wait\_loops

log\_flush\_total\_time

log\_free\_space

log\_full\_block\_writes

log\_lsn\_archived

log\_lsn\_buf\_dirty\_pages\_added

log\_lsn\_buf\_pool\_oldest\_approx

log\_lsn\_buf\_pool\_oldest\_lwm

log\_lsn\_checkpoint\_age

log\_lsn\_current

log\_lsn\_last\_checkpoint

log\_lsn\_last\_flush

log\_max\_modified\_age\_async

log\_max\_modified\_age\_sync

log\_next\_file

log\_on\_buffer\_space\_no\_waits

log\_on\_buffer\_space\_waits

log\_on\_buffer\_space\_wait\_loops

log\_on\_file\_space\_no\_waits

log\_on\_file\_space\_waits

log\_on\_file\_space\_wait\_loops

log\_on\_flush\_no\_waits

log\_on\_flush\_waits

log\_on\_flush\_wait\_loops

log\_on\_recent\_closed\_wait\_loops

log\_on\_recent\_written\_wait\_loops

log\_on\_write\_no\_waits

log\_on\_write\_waits

log\_on\_write\_wait\_loops

log\_padded

log\_partial\_block\_writes

log\_waits

log\_writer\_no\_waits

log\_writer\_on\_archiver\_waits

log\_writer\_on\_file\_space\_waits

log\_writer\_waits

log\_writer\_wait\_loops

log\_writes

log\_write\_notifier\_no\_waits

log\_write\_notifier\_waits

log\_write\_notifier\_wait\_loops

log\_write\_requests

log\_write\_to\_file\_requests\_interval

metadata\_table\_handles\_closed

metadata\_table\_handles\_opened

metadata\_table\_reference\_count

module\_cpu

module\_dblwr

module\_page\_track

os\_data\_fsyncs

os\_data\_reads

os\_data\_writes

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

lock

lock

lock

lock

lock

lock

lock

lock

lock

lock

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

log

metadata

metadata

metadata

cpu

dblwr

page\_track

os

os

os

|  |  |  |
| --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | os\_log\_bytes\_written  os\_log\_fsyncs  os\_log\_pending\_fsyncs  os\_log\_pending\_writes  os\_pending\_reads  os\_pending\_writes  page\_track\_checkpoint\_partial\_flush\_request |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | os  os  os  os  os  os  page\_track  page\_track  page\_track  page\_track  purge  purge  purge  purge  purge  purge  purge  purge  purge  sampling  sampling  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  transaction  undo  undo  undo  undo |

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

314 rows in set (0.00 sec)

page\_track\_full\_block\_writes

page\_track\_partial\_block\_writes

page\_track\_resets

purge\_del\_mark\_records

purge\_dml\_delay\_usec

purge\_invoked

purge\_resume\_count

purge\_stop\_count

purge\_truncate\_history\_count

purge\_truncate\_history\_usec

purge\_undo\_log\_pages

purge\_upd\_exist\_or\_extern\_records

sampled\_pages\_read

sampled\_pages\_skipped

trx\_active\_transactions

trx\_allocations

trx\_commits\_insert\_update

trx\_nl\_ro\_commits

trx\_on\_log\_no\_waits

trx\_on\_log\_waits

trx\_on\_log\_wait\_loops

trx\_rollbacks

trx\_rollbacks\_savepoint

trx\_rollback\_active

trx\_ro\_commits

trx\_rseg\_current\_size

trx\_rseg\_history\_len

trx\_rw\_commits

trx\_undo\_slots\_cached

trx\_undo\_slots\_used

undo\_truncate\_count

undo\_truncate\_done\_logging\_count

undo\_truncate\_start\_logging\_count

undo\_truncate\_usec

|  |  |
| --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | enabled  enabled  enabled  enabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled  enabled  disabled  disabled  disabled  disabled  disabled  disabled  disabled |

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

**Counter** **Modules**

Each counter is associated with a particular module. Module names can be used to enable, disable, or reset all counters for a particular subsystem. For example, use module\_dml to enable all counters associated with the dml subsystem.

mysql> **SET** **GLOBAL** **innodb\_monitor\_enable** **=** **module\_dml;**

mysql> **SELECT** **name,** **subsystem,** **status** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS**

**WHERE** **subsystem** **='dml';**

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

| name | subsystem | status |

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

| dml\_reads | dml | enabled |

| dml\_inserts | dml | enabled |

| dml\_deletes | dml | enabled |

| dml\_updates | dml | enabled |

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

Module names can be used with [innodb\_monitor\_enable](#_bookmark204) and related variables. Module names and corresponding SUBSYSTEM names are listed below.

• module\_adaptive\_hash (subsystem = adaptive\_hash\_index)

• module\_buffer (subsystem = buffer)

• module\_buffer\_page (subsystem = buffer\_page\_io)

• module\_compress (subsystem = compression)

• module\_ddl (subsystem = ddl)

• module\_dml (subsystem = dml)

• module\_file (subsystem = file\_system)

• module\_ibuf\_system (subsystem = change\_buffer)

• module\_icp (subsystem = icp)

• module\_index (subsystem = index)

• module\_innodb (subsystem = innodb)

• module\_lock (subsystem = lock)

• module\_log (subsystem = log)

• module\_metadata (subsystem = metadata)

• module\_os (subsystem = os)

• module\_purge (subsystem = purge)

• module\_trx (subsystem = transaction)

• module\_undo (subsystem = undo)

**Example** **15.11** **Working** **with** **INNODB\_METRICS** **Table** **Counters**

This example demonstrates enabling, disabling, and resetting a counter, and querying counter data in the INNODB\_METRICS table.

1. Create a simple InnoDB table:

mysql> **USE** **test;**

Database changed

mysql> **CREATE** **TABLE** **t1** **(c1** **INT)** **ENGINE=INNODB;**

Query OK, 0 rows affected (0.02 sec)

2. Enable the dml\_inserts counter.

mysql> **SET** **GLOBAL** **innodb\_monitor\_enable** **=** **dml\_inserts;**

Query OK, 0 rows affected (0.01 sec)

A description of the dml\_inserts counter can be found in the COMMENT column of the INNODB\_METRICS table:

mysql> **SELECT** **NAME,** **COMMENT** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts";**

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

| NAME | COMMENT |

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

| dml\_inserts | Number of rows inserted |

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

3. Query the INNODB\_METRICS table for the dml\_inserts counter data. Because no DML operations have been performed, the counter values are zero or NULL. The TIME\_ENABLED and TIME\_ELAPSED values indicate when the counter was last enabled and how many seconds have elapsed since that time.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 0

MAX\_COUNT: 0

MIN\_COUNT: NULL

AVG\_COUNT: 0

COUNT\_RESET: 0

MAX\_COUNT\_RESET: 0

MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: NULL

TIME\_ENABLED: 2014-12-04 14:18:28

TIME\_DISABLED: NULL

TIME\_ELAPSED: 28

TIME\_RESET: NULL

STATUS: enabled

TYPE: status\_counter

COMMENT: Number of rows inserted

4. Insert three rows of data into the table.

mysql> **INSERT** **INTO** **t1** **values(1);**

Query OK, 1 row affected (0.00 sec)

mysql> **INSERT** **INTO** **t1** **values(2);**

Query OK, 1 row affected (0.00 sec)

mysql> **INSERT** **INTO** **t1** **values(3);**

Query OK, 1 row affected (0.00 sec)

5. Query the INNODB\_METRICS table again for the dml\_inserts counter data. A number of counter values have now incremented including COUNT, MAX\_COUNT, AVG\_COUNT, and COUNT\_RESET. Refer to the INNODB\_METRICS table definition for descriptions of these values.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 3

MAX\_COUNT: 3

MIN\_COUNT: NULL

AVG\_COUNT: 0.046153846153846156

COUNT\_RESET: 3

MAX\_COUNT\_RESET: 3

MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: NULL

TIME\_ENABLED: 2014-12-04 14:18:28

TIME\_DISABLED: NULL

TIME\_ELAPSED: 65

TIME\_RESET: NULL

STATUS: enabled

TYPE: status\_counter

COMMENT: Number of rows inserted

6. Reset the dml\_inserts counter and query the INNODB\_METRICS table again for the dml\_inserts counter data. The %\_RESET values that were reported previously, such as COUNT\_RESET and MAX\_RESET, are set back to zero. Values such as COUNT, MAX\_COUNT, and AVG\_COUNT, which cumulatively collect data from the time the counter is enabled, are unaffected by the reset.

mysql> **SET** **GLOBAL** **innodb\_monitor\_reset** **=** **dml\_inserts;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 3

MAX\_COUNT: 3

MIN\_COUNT: NULL

AVG\_COUNT: 0.03529411764705882

COUNT\_RESET: 0

MAX\_COUNT\_RESET: 0



MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: 0

|  |  |
| --- | --- |
| TIME\_ENABLED: 2014-12-04  TIME\_DISABLED: NULL  TIME\_ELAPSED: 85  TIME\_RESET: 2014-12-04  STATUS: enabled | 14:18:28  14:19:44 |

TYPE: status\_counter

COMMENT: Number of rows inserted

7. To reset all counter values, you must first disable the counter. Disabling the counter sets the STATUS value to disabled.

mysql> **SET** **GLOBAL** **innodb\_monitor\_disable** **=** **dml\_inserts;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 3

MAX\_COUNT: 3

MIN\_COUNT: NULL

AVG\_COUNT: 0.030612244897959183

COUNT\_RESET: 0

MAX\_COUNT\_RESET: 0

MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: 0

|  |  |
| --- | --- |
| TIME\_ENABLED: 2014-12-04  TIME\_DISABLED: 2014-12-04  TIME\_ELAPSED: 98  TIME\_RESET: NULL  STATUS: disabled | 14:18:28  14:20:06 |

TYPE: status\_counter

COMMENT: Number of rows inserted

**Note**

Wildcard match is supported for counter and module names. For example, instead of specifying the full dml\_inserts counter name, you can specify dml\_i%. You can also enable, disable, or reset multiple counters or modules at once using a wildcard match. For example, specify dml\_% to enable, disable, or reset all counters that begin with dml\_.

8. After the counter is disabled, you can reset all counter values using the

[innodb\_monitor\_reset\_all](#_bookmark207) option. All values are set to zero or NULL.

mysql> **SET** **GLOBAL** **innodb\_monitor\_reset\_all** **=** **dml\_inserts;**

Query OK, 0 rows affected (0.00 sec)

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_METRICS** **WHERE** **NAME="dml\_inserts"\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

NAME: dml\_inserts

SUBSYSTEM: dml

COUNT: 0

MAX\_COUNT: NULL

MIN\_COUNT: NULL

AVG\_COUNT: NULL

COUNT\_RESET: 0

MAX\_COUNT\_RESET: NULL

MIN\_COUNT\_RESET: NULL

AVG\_COUNT\_RESET: NULL

TIME\_ENABLED: NULL

TIME\_DISABLED: NULL

TIME\_ELAPSED: NULL

TIME\_RESET: NULL

STATUS: disabled

TYPE: status\_counter

COMMENT: Number of rows inserted

**15.15.7** **InnoDB** **INFORMATION\_SCHEMA** **Temporary** **Table** **Info** **Table**

INNODB\_TEMP\_TABLE\_INFO provides information about user-created InnoDB temporary tables that are active in the InnoDB instance. It does not provide information about internal InnoDB temporary tables used by the optimizer.

mysql> **SHOW** **TABLES** **FROM** **INFORMATION\_SCHEMA** **LIKE** **'INNODB\_TEMP%';**

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

| Tables\_in\_INFORMATION\_SCHEMA (INNODB\_TEMP%) |

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

| INNODB\_TEMP\_TABLE\_INFO |

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

For the table definition, see Section 26.4.27, “The INFORMATION\_SCHEMA

INNODB\_TEMP\_TABLE\_INFO Table” .

**Example** **15.12** **INNODB\_TEMP\_TABLE\_INFO**

This example demonstrates characteristics of the INNODB\_TEMP\_TABLE\_INFO table.

1. Create a simple InnoDB temporary table:

mysql> **CREATE** **TEMPORARY** **TABLE** **t1** **(c1** **INT** **PRIMARY** **KEY)** **ENGINE=INNODB;**

2. Query INNODB\_TEMP\_TABLE\_INFO to view the temporary table metadata.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TEMP\_TABLE\_INFO\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 194

NAME: #sql7a79\_1\_0

N\_COLS: 4

SPACE: 182

The TABLE\_ID is a unique identifier for the temporary table. The NAME column displays the system-generated name for the temporary table, which is prefixed with “#sql” . The number of columns (N\_COLS) is 4 rather than 1 because InnoDB always creates three hidden table columns (DB\_ROW\_ID, DB\_TRX\_ID, and DB\_ROLL\_PTR).

3. Restart MySQL and query INNODB\_TEMP\_TABLE\_INFO.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TEMP\_TABLE\_INFO\G**

An empty set is returned because INNODB\_TEMP\_TABLE\_INFO and its data are not persisted to disk when the server is shut down.

4. Create a new temporary table.

mysql> **CREATE** **TEMPORARY** **TABLE** **t1** **(c1** **INT** **PRIMARY** **KEY)** **ENGINE=INNODB;**

5. Query INNODB\_TEMP\_TABLE\_INFO to view the temporary table metadata.

mysql> **SELECT** **\*** **FROM** **INFORMATION\_SCHEMA.INNODB\_TEMP\_TABLE\_INFO\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

TABLE\_ID: 196

NAME: #sql7b0e\_1\_0

N\_COLS: 4

SPACE: 184

The SPACE ID may be different because it is dynamically generated when the server is started.

**15.15.8** **Retrieving** **InnoDB** **Tablespace** **Metadata** **from**

**INFORMATION\_SCHEMA.FILES**

The Information Schema FILES table provides metadata about all InnoDB tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, temporary table tablespaces, and undo tablespaces (if present).



This section provides InnoDB-specific usage examples. For more information about data provided by the Information Schema FILES table, see Section 26.3.15, “The INFORMATION\_SCHEMA FILES

Table” .

**Note**

The INNODB\_TABLESPACES and INNODB\_DATAFILES tables also provide metadata about InnoDB tablespaces, but data is limited to file-per-table, general, and undo tablespaces.

This query retrieves metadata about the InnoDB system tablespace from fields of the Information Schema FILES table that are pertinent to InnoDB tablespaces. FILES columns that are not relevant to InnoDB always return NULL, and are excluded from the query.

mysql> **SELECT** **FILE\_ID,** **FILE\_NAME,** **FILE\_TYPE,** **TABLESPACE\_NAME,** **FREE\_EXTENTS,**

**TOTAL\_EXTENTS,** **EXTENT\_SIZE,** **INITIAL\_SIZE,** **MAXIMUM\_SIZE,** **AUTOEXTEND\_SIZE,** **DATA\_FREE,** **STATUS** **ENGI**

**FROM** **INFORMATION\_SCHEMA.FILES** **WHERE** **TABLESPACE\_NAME** **LIKE** **'innodb\_system'** **\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_ID: 0

FILE\_NAME: ./ibdata1

FILE\_TYPE: TABLESPACE

TABLESPACE\_NAME: innodb\_system

FREE\_EXTENTS: 0

TOTAL\_EXTENTS: 12

EXTENT\_SIZE: 1048576

INITIAL\_SIZE: 12582912

MAXIMUM\_SIZE: NULL

AUTOEXTEND\_SIZE: 67108864

DATA\_FREE: 4194304

ENGINE: NORMAL

This query retrieves the FILE\_ID (equivalent to the space ID) and the FILE\_NAME (which includes path information) for InnoDB file-per-table and general tablespaces. File-per-table and general tablespaces have a .ibd file extension.

mysql> **SELECT** **FILE\_ID,** **FILE\_NAME** **FROM** **INFORMATION\_SCHEMA.FILES**

**WHERE** **FILE\_NAME** **LIKE** **'%** **.ibd%'** **ORDER** **BY** **FILE\_ID;**

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

| FILE\_ID | FILE\_NAME |

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

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  | | 2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  23  26 | |  |  |  |  |  |  |  |  |  |  |  |  |  | | ./mysql/plugin .ibd  ./mysql/servers .ibd  ./mysql/help\_topic .ibd  ./mysql/help\_category .ibd  ./mysql/help\_relation .ibd  ./mysql/help\_keyword .ibd  ./mysql/time\_zone\_name .ibd  ./mysql/time\_zone .ibd  ./mysql/time\_zone\_transition .ibd  ./mysql/time\_zone\_transition\_type .ibd  ./mysql/time\_zone\_leap\_second .ibd  ./mysql/innodb\_table\_stats .ibd  ./mysql/innodb\_index\_stats .ibd  ./mysql/slave\_relay\_log\_info .ibd  ./mysql/slave\_master\_info .ibd  ./mysql/slave\_worker\_info .ibd  ./mysql/gtid\_executed .ibd  ./mysql/server\_cost .ibd  ./mysql/engine\_cost .ibd  ./sys/sys\_config .ibd  ./test/t1 .ibd  /home/user/test/test/t2 ibd | |  |  |  |  |  |  |  |  |  |  |  |  |  | |

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

This query retrieves the FILE\_ID and FILE\_NAME for the InnoDB global temporary tablespace. Global temporary tablespace file names are prefixed by ibtmp.

mysql> **SELECT** **FILE\_ID,** **FILE\_NAME** **FROM** **INFORMATION\_SCHEMA.FILES**

**WHERE** **FILE\_NAME** **LIKE** **'%ibtmp%';**

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

| FILE\_ID | FILE\_NAME |

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

|  |  |
| --- | --- |
| | | 22 | ./ibtmp1 | |

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

Similarly, InnoDB undo tablespace file names are prefixed by undo. The following query returns the FILE\_ID and FILE\_NAME for InnoDB undo tablespaces.

mysql> **SELECT** **FILE\_ID,** **FILE\_NAME** **FROM** **INFORMATION\_SCHEMA.FILES**

**WHERE** **FILE\_NAME** **LIKE** **'%undo%';**

**15.16** **InnoDB** **Integration** **with** **MySQL** **Performance** **Schema**

This section provides a brief introduction to InnoDB integration with Performance Schema. For comprehensive Performance Schema documentation, see Chapter 27, *MySQL* *Performance* *Schema*.

You can profile certain internal InnoDB operations using the MySQL Performance Schema feature. This type of tuning is primarily for expert users who evaluate optimization strategies to overcome performance bottlenecks. DBAs can also use this feature for capacity planning, to see whether their typical workload encounters any performance bottlenecks with a particular combination of CPU, RAM, and disk storage; and if so, to judge whether performance can be improved by increasing the capacity of some part of the system.

To use this feature to examine InnoDB performance:

• You must be generally familiar with how to use the Performance Schema feature. For example, you should know how enable instruments and consumers, and how to query performance\_schema tables to retrieve data. For an introductory overview, see Section 27.1, “Performance Schema Quick Start” .

• You should be familiar with Performance Schema instruments that are available for InnoDB. To view InnoDB-related instruments, you can query the setup\_instruments table for instrument names that contain 'innodb'.

mysql> **SELECT** **\***

**FROM** **performance\_schema.setup\_instruments**

**WHERE** **NAME** **LIKE** **'%innodb%';**

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

|  |  |
| --- | --- |
| | NAME | | ENABLED | TIMED | |

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

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | wait/synch/mutex/innodb/commit\_cond\_mutex  | wait/synch/mutex/innodb/innobase\_share\_mutex  | wait/synch/mutex/innodb/autoinc\_mutex  | wait/synch/mutex/innodb/buf\_pool\_mutex  | wait/synch/mutex/innodb/buf\_pool\_zip\_mutex  | wait/synch/mutex/innodb/cache\_last\_read\_mutex  | wait/synch/mutex/innodb/dict\_foreign\_err\_mutex  | wait/synch/mutex/innodb/dict\_sys\_mutex  | wait/synch/mutex/innodb/recalc\_pool\_mutex  ...  | wait/io/file/innodb/innodb\_data\_file  | wait/io/file/innodb/innodb\_log\_file  | wait/io/file/innodb/innodb\_temp\_file  | stage/innodb/alter table (end)  | stage/innodb/alter table (flush)  | stage/innodb/alter table (insert)  | stage/innodb/alter table (log apply index)  | stage/innodb/alter table (log apply table)  | stage/innodb/alter table (merge sort)  | stage/innodb/alter table (read PK and internal sort)  | stage/innodb/buffer pool load  | memory/innodb/buf\_buf\_pool  | memory/innodb/dict\_stats\_bg\_recalc\_pool\_t  | memory/innodb/dict\_stats\_index\_map\_t  | memory/innodb/dict\_stats\_n\_diff\_on\_level  | memory/innodb/other  | memory/innodb/row\_log\_buf  | memory/innodb/row\_merge\_sort | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | NO  NO  NO  NO  NO  NO  NO  NO  NO  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  NO  NO  NO  NO  NO  NO  NO | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | NO  NO  NO  NO  NO  NO  NO  NO  NO  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  YES  NO  NO  NO  NO  NO  NO  NO | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | memory/innodb/std  | memory/innodb/sync\_debug\_latches  | memory/innodb/trx\_sys\_t::rw\_trx\_ids  ... | |  |  | | NO  NO  NO | |  |  | | NO  NO  NO | |  |  | |

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

155 rows in set (0.00 sec)

For additional information about the instrumented InnoDB objects, you can query Performance Schema instances tables, which provide additional information about instrumented objects. Instance tables relevant to InnoDB include:

• The mutex\_instances table

• The rwlock\_instances table

• The cond\_instances table

• The file\_instances table

**Note**

Mutexes and RW-locks related to the InnoDB buffer pool are not included in this coverage; the same applies to the output of the SHOW ENGINE INNODB MUTEX command.

For example, to view information about instrumented InnoDB file objects seen by the Performance Schema when executing file I/O instrumentation, you might issue the following query:

mysql> **SELECT** **\***

**FROM** **performance\_schema.file\_instances**

**WHERE** **EVENT\_NAME** **LIKE** **'%innodb%'\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_NAME: /home/dtprice/mysql-8 .0/data/ibdata1

EVENT\_NAME: wait/io/file/innodb/innodb\_data\_file

OPEN\_COUNT: 3

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 2. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_NAME: /home/dtprice/mysql-8 .0/data/#ib\_16384\_0 .dblwr

EVENT\_NAME: wait/io/file/innodb/innodb\_dblwr\_file

OPEN\_COUNT: 2

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 3. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

FILE\_NAME: /home/dtprice/mysql-8 .0/data/#ib\_16384\_1 .dblwr

EVENT\_NAME: wait/io/file/mysql-8.0/innodb\_dblwr\_file

OPEN\_COUNT: 2

...

• You should be familiar with performance\_schema tables that store InnoDB event data. Tables relevant to InnoDB-related events include:

• The Wait Event tables, which store wait events.

• The Summary tables, which provide aggregated information for terminated events over time. Summary tables include file I/O summary tables, which aggregate information about I/O

operations.

• Stage Event tables, which store event data for InnoDB ALTER TABLE and buffer pool load operations. For more information, see [Section 15.16.1, “Monitoring ALTER TABLE Progress for](#_bookmark68) [InnoDB Tables Using Performance Schema”](#_bookmark68) , and Monitoring Buffer Pool Load Progress Using Performance Schema.

If you are only interested in InnoDB-related objects, use the clause WHERE EVENT\_NAME LIKE '%innodb%' or WHERE NAME LIKE '%innodb%' (as required) when querying these tables.

**15.16.1** **Monitoring** **ALTER** **TABLE** **Progress** **for** **InnoDB** **Tables** **Using**

**Performance** **Schema**



**WHERE** **NAME** **LIKE** **'stage/innodb/alter%';**

Query OK, 7 rows affected (0 .00 sec)

Rows matched: 7 Changed: 7 Warnings: 0

You can monitor ALTER TABLE progress for InnoDB tables using Performance Schema.

There are seven stage events that represent different phases of ALTER TABLE. Each stage event reports a running total of WORK\_COMPLETED and WORK\_ESTIMATED for the overall ALTER TABLE operation as it progresses through its different phases. WORK\_ESTIMATED is calculated using a formula that takes into account all of the work that ALTER TABLE performs, and may be revised during ALTER TABLE processing. WORK\_COMPLETED and WORK\_ESTIMATED values are an abstract representation of all of the work performed by ALTER TABLE.

In order of occurrence, ALTER TABLE stage events include:

• stage/innodb/alter table (read PK and internal sort): This stage is active

when ALTER TABLE is in the reading-primary-key phase. It starts with WORK\_COMPLETED=0 and WORK\_ESTIMATED set to the estimated number of pages in the primary key. When the stage is completed, WORK\_ESTIMATED is updated to the actual number of pages in the primary key.

• stage/innodb/alter table (merge sort): This stage is repeated for each index added by the ALTER TABLE operation.

• stage/innodb/alter table (insert): This stage is repeated for each index added by the ALTER TABLE operation.

• stage/innodb/alter table (log apply index): This stage includes the application of DML log generated while ALTER TABLE was running.

• stage/innodb/alter table (flush): Before this stage begins, WORK\_ESTIMATED is updated with a more accurate estimate, based on the length of the flush list.

• stage/innodb/alter table (log apply table): This stage includes the application of concurrent DML log generated while ALTER TABLE was running. The duration of this phase depends on the extent of table changes. This phase is instant if no concurrent DML was run on the table.

• stage/innodb/alter table (end): Includes any remaining work that appeared after the flush phase, such as reapplying DML that was executed on the table while ALTER TABLE was running.

**Note**

InnoDB ALTER TABLE stage events do not currently account for the addition of spatial indexes.

**ALTER** **TABLE** **Monitoring** **Example** **Using** **Performance** **Schema**

The following example demonstrates how to enable the stage/innodb/alter table% stage event instruments and related consumer tables to monitor ALTER TABLE progress. For information about Performance Schema stage event instruments and related consumers, see Section 27.12.5, “Performance Schema Stage Event Tables” .

1. Enable the stage/innodb/alter% instruments:

mysql> **UPDATE** **performance\_schema.setup\_instruments**

**SET** **ENABLED** **=** **'YES'**

2. Enable the stage event consumer tables, which include events\_stages\_current, events\_stages\_history, and events\_stages\_history\_long.

mysql> **UPDATE** **performance\_schema.setup\_consumers**

**SET** **ENABLED** **=** **'YES'**

**WHERE** **NAME** **LIKE** **'%stages%';**

**FROM** **performance\_schema** **.events\_stages\_current;**

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

| EVENT\_NAME | WORK\_COMPLETED | WORK\_ESTIMATED |

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

| stage/innodb/alter table (read PK and internal sort) | 280 | 1245 |

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

1 row in set (0.01 sec)

Query OK, 3 rows affected (0.00 sec)

Rows matched: 3 Changed: 3 Warnings: 0

3. Run an ALTER TABLE operation. In this example, a middle\_name column is added to the employees table of the employees sample database.

mysql> **ALTER** **TABLE** **employees** **.employees** **ADD** **COLUMN** **middle\_name** **varchar(14)** **AFTER** **first\_name;**

Query OK, 0 rows affected (9.27 sec)

Records: 0 Duplicates: 0 Warnings: [0](#_bookmark250)

4. Check the progress of the ALTER TABLE operation by querying the Performance Schema events\_stages\_current table. The stage event shown differs depending on which ALTER TABLE phase is currently in progress. The WORK\_COMPLETED column shows the work completed. The WORK\_ESTIMATED column provides an estimate of the remaining work.

mysql> **SELECT** **EVENT\_NAME,** **WORK\_COMPLETED,** **WORK\_ESTIMATED**

The events\_stages\_current table returns an empty set if the ALTER TABLE operation has completed. In this case, you can check the events\_stages\_history table to view event data for the completed operation. For example:

mysql> **SELECT** **EVENT\_NAME,** **WORK\_COMPLETED,** **WORK\_ESTIMATED**

**FROM** **performance\_schema** **.events\_stages\_history;**

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

| EVENT\_NAME | WORK\_COMPLETED | WORK\_ESTIMATED |

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

| stage/innodb/alter table (read PK and internal sort) |

|

|

|

|

|

|

|

|

|

|

886

1213

1597

1597

1981

1213

1213

1597

1597

1981

| stage/innodb/alter table (flush) |

| stage/innodb/alter table (log apply table) |

| stage/innodb/alter table (end) |

| stage/innodb/alter table (log apply table) |

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

5 rows in set (0.00 sec)

As shown above, the WORK\_ESTIMATED value was revised during ALTER TABLE processing. The estimated work after completion of the initial stage is 1213. When ALTER TABLE processing completed, WORK\_ESTIMATED was set to the actual value, which is 1981.

**15.16.2** **Monitoring** **InnoDB** **Mutex** **Waits** **Using** **Performance** **Schema**

A mutex is a synchronization mechanism used in the code to enforce that only one thread at a given time can have access to a common resource. When two or more threads executing in the server need to access the same resource, the threads compete against each other. The first thread to obtain a lock on the mutex causes the other threads to wait until the lock is released.

For InnoDB mutexes that are instrumented, mutex waits can be monitored using Performance Schema. Wait event data collected in Performance Schema tables can help identify mutexes with the most waits or the greatest total wait time, for example.

The following example demonstrates how to enable InnoDB mutex wait instruments, how to enable associated consumers, and how to query wait event data.

1. To view available InnoDB mutex wait instruments, query the Performance Schema setup\_instruments table. All InnoDB mutex wait instruments are disabled by default.

mysql> **SELECT** **\***

**FROM** **performance\_schema** **.setup\_instruments**

**WHERE** **NAME** **LIKE** **'%wait/synch/mutex/innodb%';**

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

wait/synch/mutex/innodb/commit\_cond\_mutex |

wait/synch/mutex/innodb/innobase\_share\_mutex |

wait/synch/mutex/innodb/autoinc\_mutex |

wait/synch/mutex/innodb/autoinc\_persisted\_mutex |

wait/synch/mutex/innodb/buf\_pool\_flush\_state\_mutex |

wait/synch/mutex/innodb/buf\_pool\_LRU\_list\_mutex |

wait/synch/mutex/innodb/buf\_pool\_free\_list\_mutex |

wait/synch/mutex/innodb/buf\_pool\_zip\_free\_mutex |

wait/synch/mutex/innodb/buf\_pool\_zip\_hash\_mutex |

wait/synch/mutex/innodb/buf\_pool\_zip\_mutex |

wait/synch/mutex/innodb/cache\_last\_read\_mutex |

wait/synch/mutex/innodb/dict\_foreign\_err\_mutex |

wait/synch/mutex/innodb/dict\_persist\_dirty\_tables\_mutex |

wait/synch/mutex/innodb/dict\_sys\_mutex |

wait/synch/mutex/innodb/recalc\_pool\_mutex |

wait/synch/mutex/innodb/fil\_system\_mutex |

wait/synch/mutex/innodb/flush\_list\_mutex |

wait/synch/mutex/innodb/fts\_bg\_threads\_mutex |

wait/synch/mutex/innodb/fts\_delete\_mutex |

wait/synch/mutex/innodb/fts\_optimize\_mutex |

wait/synch/mutex/innodb/fts\_doc\_id\_mutex |

wait/synch/mutex/innodb/log\_flush\_order\_mutex |

wait/synch/mutex/innodb/hash\_table\_mutex |

wait/synch/mutex/innodb/ibuf\_bitmap\_mutex |

wait/synch/mutex/innodb/ibuf\_mutex |

wait/synch/mutex/innodb/ibuf\_pessimistic\_insert\_mutex |

wait/synch/mutex/innodb/log\_sys\_mutex |

wait/synch/mutex/innodb/log\_sys\_write\_mutex |

wait/synch/mutex/innodb/mutex\_list\_mutex |

wait/synch/mutex/innodb/page\_zip\_stat\_per\_index\_mutex |

wait/synch/mutex/innodb/purge\_sys\_pq\_mutex |

wait/synch/mutex/innodb/recv\_sys\_mutex |

wait/synch/mutex/innodb/recv\_writer\_mutex |

wait/synch/mutex/innodb/redo\_rseg\_mutex |

wait/synch/mutex/innodb/noredo\_rseg\_mutex |

wait/synch/mutex/innodb/rw\_lock\_list\_mutex |

wait/synch/mutex/innodb/rw\_lock\_mutex |

wait/synch/mutex/innodb/srv\_dict\_tmpfile\_mutex |

wait/synch/mutex/innodb/srv\_innodb\_monitor\_mutex |

wait/synch/mutex/innodb/srv\_misc\_tmpfile\_mutex |

wait/synch/mutex/innodb/srv\_monitor\_file\_mutex |

wait/synch/mutex/innodb/buf\_dblwr\_mutex |

wait/synch/mutex/innodb/trx\_undo\_mutex |

wait/synch/mutex/innodb/trx\_pool\_mutex |

wait/synch/mutex/innodb/trx\_pool\_manager\_mutex |

wait/synch/mutex/innodb/srv\_sys\_mutex |

wait/synch/mutex/innodb/lock\_mutex |

wait/synch/mutex/innodb/lock\_wait\_mutex |

wait/synch/mutex/innodb/trx\_mutex |

wait/synch/mutex/innodb/srv\_threads\_mutex |

wait/synch/mutex/innodb/rtr\_active\_mutex |

wait/synch/mutex/innodb/rtr\_match\_mutex |

wait/synch/mutex/innodb/rtr\_path\_mutex |

wait/synch/mutex/innodb/rtr\_ssn\_mutex |

wait/synch/mutex/innodb/trx\_sys\_mutex |

wait/synch/mutex/innodb/zip\_pad\_mutex |

wait/synch/mutex/innodb/master\_key\_id\_mutex |

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

NO

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

| NAME

| ENABLED | TIMED |

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

|  |  |  |
| --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO  NO | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |

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

2. Some InnoDB mutex instances are created at server startup and are only instrumented if the associated instrument is also enabled at server startup. To ensure that all InnoDB mutex instances are instrumented and enabled, add the following performance-schema-instrument rule to your MySQL configuration file:

performance-schema-instrument='wait/synch/mutex/innodb/%=ON'

If you do not require wait event data for all InnoDB mutexes, you can disable specific instruments by adding additional performance-schema-instrument rules to your MySQL configuration file.



**WHERE** **name** **like** **'events\_waits%';**

Query OK, 3 rows affected (0.00 sec)

Rows matched: 3 Changed: 3 Warnings: 0

For example, to disable InnoDB mutex wait event instruments related to full-text search, add the following rule:

performance-schema-instrument='wait/synch/mutex/innodb/fts%=OFF'

**Note**

Rules with a longer prefix such as wait/synch/mutex/innodb/fts%

take precedence over rules with shorter prefixes such as wait/synch/ mutex/innodb/%.

After adding the performance-schema-instrument rules to your configuration file, restart the server. All the InnoDB mutexes except for those related to full text search are enabled. To verify, query the setup\_instruments table. The ENABLED and TIMED columns should be set to YES for the instruments that you enabled.

mysql> **SELECT** **\***

**FROM** **performance\_schema** **.setup\_instruments**

**WHERE** **NAME** **LIKE** **'%wait/synch/mutex/innodb%';**

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

| ENABLED | TIMED |

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

|

|

|

...

|

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

49 rows in set (0.00 sec)

| wait/synch/mutex/innodb/commit\_cond\_mutex

| wait/synch/mutex/innodb/innobase\_share\_mutex

| wait/synch/mutex/innodb/autoinc\_mutex

| wait/synch/mutex/innodb/master\_key\_id\_mutex

YES

YES

YES

YES

YES

YES

| NAME

|

|

|

|

|

|

|

|

YES

YES

3. Enable wait event consumers by updating the setup\_consumers table. Wait event consumers are disabled by default.

mysql> **UPDATE** **performance\_schema.setup\_consumers**

**SET** **enabled** **=** **'YES'**

You can verify that wait event consumers are enabled by querying the setup\_consumers table. The events\_waits\_current, events\_waits\_history, and events\_waits\_history\_long consumers should be enabled.

mysql> **SELECT** **\*** **FROM** **performance\_schema** **.setup\_consumers;**

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

| NAME | ENABLED |

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

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

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

15 rows in set (0.00 sec)

events\_stages\_current

events\_stages\_history

events\_stages\_history\_long

events\_statements\_current

events\_statements\_history

events\_statements\_history\_long

events\_transactions\_current

events\_transactions\_history

events\_transactions\_history\_long

events\_waits\_current

events\_waits\_history

events\_waits\_history\_long

global\_instrumentation

thread\_instrumentation

statements\_digest

NO

NO

NO

YES

YES

NO

YES

YES

NO

YES

YES

YES

YES

YES

YES

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

|

4. Once instruments and consumers are enabled, run the workload that you want to monitor. In this example, the mysqlslap load emulation client is used to simulate a workload.

$> **./mysqlslap** **--auto-generate-sql** **--concurrency=100** **--iterations=10**

**--number-of-queries=1000** **--number-char-cols=6** **--number-int-cols=6;**

5. Query the wait event data. In this example, wait event data is queried from the events\_waits\_summary\_global\_by\_event\_name table which aggregates data found in the events\_waits\_current, events\_waits\_history, and events\_waits\_history\_long tables. Data is summarized by event name (EVENT\_NAME), which is the name of the instrument that produced the event. Summarized data includes:

• COUNT\_STAR

The number of summarized wait events.

• SUM\_TIMER\_WAIT

The total wait time of the summarized timed wait events.

• MIN\_TIMER\_WAIT

The minimum wait time of the summarized timed wait events.

• AVG\_TIMER\_WAIT

The average wait time of the summarized timed wait events.

• MAX\_TIMER\_WAIT

The maximum wait time of the summarized timed wait events.

The following query returns the instrument name (EVENT\_NAME), the number of wait events (COUNT\_STAR), and the total wait time for the events for that instrument (SUM\_TIMER\_WAIT). Because waits are timed in picoseconds (trillionths of a second) by default, wait times are divided by 1000000000 to show wait times in milliseconds. Data is presented in descending order, by the number of summarized wait events (COUNT\_STAR). You can adjust the ORDER BY clause to order the data by total wait time.

mysql> **SELECT** **EVENT\_NAME,** **COUNT\_STAR,** **SUM\_TIMER\_WAIT/1000000000** **SUM\_TIMER\_WAIT\_MS**

**FROM** **performance\_schema** **.events\_waits\_summary\_global\_by\_event\_name**

**WHERE** **SUM\_TIMER\_WAIT** **>** **0** **AND** **EVENT\_NAME** **LIKE** **'wait/synch/mutex/innodb/%'**

**ORDER** **BY** **COUNT\_STAR** **DESC;**

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

|  |  |
| --- | --- |
| | EVENT\_NAME | | COUNT\_STAR | SUM\_TIMER\_WAIT\_MS | |

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

|  |  |  |  |
| --- | --- | --- | --- |
| 201111  62244  48238  46113  35134  34872  17805  14912  10634  8538  5961  4885  4364  3212  3178  2495  1318  1250  951  670 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | 23.4719  9.6426  3.1135  2.0434  1068.1588  1039.2589  1526.0490  1606.7348  1.1424  0.1960  0.6473  8821.7496  0.2077  0.2650  0.2349  0.1310  0.2161  0.0893  0.0918  0.0942 | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |

|  |  |  |
| --- | --- | --- |
| |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | | wait/synch/mutex/innodb/trx\_mutex  wait/synch/mutex/innodb/fil\_system\_mutex  wait/synch/mutex/innodb/redo\_rseg\_mutex  wait/synch/mutex/innodb/log\_sys\_mutex  wait/synch/mutex/innodb/trx\_sys\_mutex  wait/synch/mutex/innodb/lock\_mutex  wait/synch/mutex/innodb/log\_sys\_write\_mutex  wait/synch/mutex/innodb/dict\_sys\_mutex  wait/synch/mutex/innodb/trx\_undo\_mutex  wait/synch/mutex/innodb/rw\_lock\_list\_mutex  wait/synch/mutex/innodb/buf\_pool\_free\_list\_mutex  wait/synch/mutex/innodb/trx\_pool\_mutex  wait/synch/mutex/innodb/buf\_pool\_LRU\_list\_mutex  wait/synch/mutex/innodb/innobase\_share\_mutex  wait/synch/mutex/innodb/flush\_list\_mutex  wait/synch/mutex/innodb/trx\_pool\_manager\_mutex  wait/synch/mutex/innodb/buf\_pool\_flush\_state\_mutex  wait/synch/mutex/innodb/log\_flush\_order\_mutex  wait/synch/mutex/innodb/buf\_dblwr\_mutex  wait/synch/mutex/innodb/recalc\_pool\_mutex | |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  | |



|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| | | wait/synch/mutex/innodb/dict\_persist\_dirty\_tables\_mutex | | 345 | | | 0.0414 | | |
| | | wait/synch/mutex/innodb/lock\_wait\_mutex | | 303 | | | 0.1565 | | |
| | | wait/synch/mutex/innodb/autoinc\_mutex | | 196 | | | 0.0213 | | |
| | | wait/synch/mutex/innodb/autoinc\_persisted\_mutex | | 196 | | | 0.0175 | | |
| | | wait/synch/mutex/innodb/purge\_sys\_pq\_mutex | | 117 | | | 0.0308 | | |
| | | wait/synch/mutex/innodb/srv\_sys\_mutex | | 94 | | | 0.0077 | | |
| | | wait/synch/mutex/innodb/ibuf\_mutex | | 22 | | | 0.0086 | | |
| | | wait/synch/mutex/innodb/recv\_sys\_mutex | | 12 | | | 0.0008 | | |
| | | wait/synch/mutex/innodb/srv\_innodb\_monitor\_mutex | | 4 | | | 0.0009 | | |
| | | wait/synch/mutex/innodb/recv\_writer\_mutex | | 1 | | | 0.0005 | | |

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

**Note**

The preceding result set includes wait event data produced during the startup process. To exclude this data, you can truncate the events\_waits\_summary\_global\_by\_event\_name table immediately after startup and before running your workload. However, the truncate operation itself may produce a negligible amount wait event data.

mysql> **TRUNCATE** **performance\_schema.events\_waits\_summary\_global\_by\_event\_name;**

**15.17** **InnoDB** **Monitors**

InnoDB monitors provide information about the InnoDB internal state. This information is useful for performance tuning.

**15.17.1** **InnoDB** **Monitor** **Types**

There are two types of InnoDB monitor:

• The standard InnoDB Monitor displays the following types of information:

• Work done by the main background thread

• Semaphore waits

• Data about the most recent foreign key and deadlock errors

• Lock waits for transactions

• Table and record locks held by active transactions

• Pending I/O operations and related statistics

• Insert buffer and adaptive hash index statistics

• Redo log data

• Buffer pool statistics

• Row operation data

• The InnoDB Lock Monitor prints additional lock information as part of the standard InnoDB Monitor output.

**15.17.2** **Enabling** **InnoDB** **Monitors**

When InnoDB monitors are enabled for periodic output, InnoDB writes the output to mysqld server standard error output (stderr) every 15 seconds, approximately.

InnoDB sends the monitor output to stderr rather than to stdout or fixed-size memory buffers to avoid potential buffer overflows.



On Windows, stderr is directed to the default log file unless configured otherwise. If you want to direct the output to the console window rather than to the error log, start the server from a command prompt in a console window with the --console option. For more information, see Default Error Log Destination on Windows.

On Unix and Unix-like systems, stderr is typically directed to the terminal unless configured otherwise. For more information, see Default Error Log Destination on Unix and Unix-Like Systems.

InnoDB monitors should only be enabled when you actually want to see monitor information because output generation causes some performance decrement. Also, if monitor output is directed to the error log, the log may become quite large if you forget to disable the monitor later.

**Note**

To assist with troubleshooting, InnoDB temporarily enables standard InnoDB Monitor output under certain conditions. For more information, see Section 15.21, “InnoDB Troubleshooting” .

InnoDB monitor output begins with a header containing a timestamp and the monitor name. For example:

=====================================

2014-10-16 18:37:29 0x7fc2a95c1700 INNODB MONITOR OUTPUT

=====================================

The header for the standard InnoDB Monitor ( INNODB MONITOR OUTPUT) is also used for the Lock Monitor because the latter produces the same output with the addition of extra lock information.

The [innodb\_status\_output](#_bookmark231) and [innodb\_status\_output\_locks](#_bookmark232) system variables are used to enable the standard InnoDB Monitor and InnoDB Lock Monitor.

The PROCESS privilege is required to enable or disable InnoDB Monitors.

**Enabling** **the** **Standard** **InnoDB** **Monitor**

Enable the standard InnoDB Monitor by setting the [innodb\_status\_output](#_bookmark231) system variable to ON. SET GLOBAL innodb\_status\_output=ON;

To disable the standard InnoDB Monitor, set [innodb\_status\_output](#_bookmark231) to OFF.

When you shut down the server, the [innodb\_status\_output](#_bookmark231) variable is set to the default OFF value.

**Enabling** **the** **InnoDB** **Lock** **Monitor**

InnoDB Lock Monitor data is printed with the InnoDB Standard Monitor output. Both the InnoDB Standard Monitor and InnoDB Lock Monitor must be enabled to have InnoDB Lock Monitor data printed periodically.

To enable the InnoDB Lock Monitor, set the [innodb\_status\_output\_locks](#_bookmark232) system variable to ON. Both the InnoDB standard Monitor and InnoDB Lock Monitor must be enabled to have InnoDB Lock Monitor data printed periodically:

SET GLOBAL innodb\_status\_output=ON;

SET GLOBAL innodb\_status\_output\_locks=ON;

To disable the InnoDB Lock Monitor, set [innodb\_status\_output\_locks](#_bookmark232) to OFF. Set [innodb\_status\_output](#_bookmark231) to OFF to also disable the InnoDB Standard Monitor.

When you shut down the server, the [innodb\_status\_output](#_bookmark231) and

[innodb\_status\_output\_locks](#_bookmark232) variables are set to the default OFF value.



**Note**

To enable the InnoDB Lock Monitor for SHOW ENGINE INNODB STATUS output, you are only required to enable [innodb\_status\_output\_locks](#_bookmark232).

**Obtaining** **Standard** **InnoDB** **Monitor** **Output** **On** **Demand**

As an alternative to enabling the standard InnoDB Monitor for periodic output, you can obtain standard InnoDB Monitor output on demand using the SHOW ENGINE INNODB STATUS SQL statement, which fetches the output to your client program. If you are using the mysql interactive client, the output is more readable if you replace the usual semicolon statement terminator with \G:

mysql> **SHOW** **ENGINE** **INNODB** **STATUS\G**

SHOW ENGINE INNODB STATUS output also includes InnoDB Lock Monitor data if the InnoDB Lock Monitor is enabled.

**Directing** **Standard** **InnoDB** **Monitor** **Output** **to** **a** **Status** **File**

Standard InnoDB Monitor output can be enabled and directed to a status file by specifying the -- innodb-status-file option at startup. When this option is used, InnoDB creates a file named innodb\_status.*pid* in the data directory and writes output to it every 15 seconds, approximately.

InnoDB removes the status 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 output generation can affect performance, and the innodb\_status.*pid* file can become quite large over time.

**15.17.3** **InnoDB** **Standard** **Monitor** **and** **Lock** **Monitor** **Output**

The Lock Monitor is the same as the Standard Monitor except that it includes additional lock information. Enabling either monitor for periodic output turns on the same output stream, but the stream includes extra information if the Lock Monitor is enabled. For example, if you enable the Standard Monitor and Lock Monitor, that turns on a single output stream. The stream includes extra lock information until you disable the Lock Monitor.

Standard Monitor output is limited to 1MB when produced using the SHOW ENGINE INNODB STATUS

statement. This limit does not apply to output written to server standard error output (stderr). Example Standard Monitor output:

mysql> **SHOW** **ENGINE** **INNODB** **STATUS\G**

\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\* 1. row \*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*

Type: InnoDB

Name:

Status:

=====================================

2018-04-12 15:14:08 0x7f971c063700 INNODB MONITOR OUTPUT

=====================================

Per second averages calculated from the last 4 seconds

-----------------

BACKGROUND THREAD

-----------------

srv\_master\_thread loops: 15 srv\_active, 0 srv\_shutdown, 1122 srv\_idle

srv\_master\_thread log flush and writes: 0

----------

SEMAPHORES

----------

OS WAIT ARRAY INFO: reservation count 24

OS WAIT ARRAY INFO: signal count 24

RW-shared spins 4, rounds 8, OS waits 4

RW-excl spins 2, rounds 60, OS waits 2

RW-sx spins 0, rounds 0, OS waits 0

Spin rounds per wait: 2.00 RW-shared, 30.00 RW-excl, 0.00 RW-sx

------------------------

LATEST FOREIGN KEY ERROR

------------------------

2018-04-12 14:57:24 0x7f97a9c91700 Transaction:

TRANSACTION 7717, ACTIVE 0 sec inserting

mysql tables in use 1, locked 1

4 lock struct(s), heap size 1136, 3 row lock(s), undo log entries 3

MySQL thread id 8, OS thread handle 140289365317376, query id 14 localhost root update

INSERT INTO child VALUES (NULL, 1), (NULL, 2), (NULL, 3), (NULL, 4), (NULL, 5), (NULL, 6)

Foreign key constraint fails for table `test`.`child`:

,

CONSTRAINT `child\_ibfk\_1` FOREIGN KEY (`parent\_id`) REFERENCES `parent` (`id`) ON DELETE

CASCADE ON UPDATE CASCADE

Trying to add in child table, in index par\_ind tuple:

DATA TUPLE: 2 fields;

0: len 4; hex 80000003; asc ;;

1: len 4; hex 80000003; asc ;;

But in parent table `test`.`parent`, in index PRIMARY,

the closest match we can find is record:

PHYSICAL RECORD: n\_fields 3; compact format; info bits 0

0: len 4; hex 80000004; asc ;;

1: len 6; hex 000000001e19; asc ;;

2: len 7; hex 81000001110137; asc 7;;

------------

TRANSACTIONS

------------

Trx id counter 7748

Purge done for trx's n:o < 7747 undo n:o < 0 state: running but idle

History list length 19

LIST OF TRANSACTIONS FOR EACH SESSION:

---TRANSACTION 421764459790000, not started

0 lock struct(s), heap size 1136, 0 row lock(s)

---TRANSACTION 7747, ACTIVE 23 sec starting index read

mysql tables in use 1, locked 1

LOCK WAIT 2 lock struct(s), heap size 1136, 1 row lock(s)

MySQL thread id 9, OS thread handle 140286987249408, query id 51 localhost root updating

DELETE FROM t WHERE i = 1

------- TRX HAS BEEN WAITING 23 SEC FOR THIS LOCK TO BE GRANTED:

RECORD LOCKS space id 4 page no 4 n bits 72 index GEN\_CLUST\_INDEX of table `test`.`t`

trx id 7747 lock\_mode X waiting

Record lock, heap no 3 PHYSICAL RECORD: n\_fields 4; compact format; info bits 0

0: len 6; hex 000000000202; asc

1: len 6; hex 000000001e41; asc

2: len 7; hex 820000008b0110; asc

3: len 4; hex 80000001; asc ;;

------------------

;;

A;;

;;

TABLE LOCK table `test`.`t` trx id 7747 lock mode IX

RECORD LOCKS space id 4 page no 4 n bits 72 index GEN\_CLUST\_INDEX of table `test` . `t`

trx id 7747 lock\_mode X waiting

Record lock, heap no 3 PHYSICAL RECORD: n\_fields 4; compact format; info bits 0

0: len

1: len

2: len

3: len

6; hex

6; hex

7; hex

4; hex

000000000202; asc

000000001e41; asc

820000008b0110; asc

80000001; asc ;;

;;

A;;

;;

--------

FILE I/O

--------

I/O thread 0 state: waiting for i/o request (insert buffer thread)

I/O thread 1 state: waiting for i/o request (log thread)

I/O thread 2 state: waiting for i/o request (read thread)

I/O thread 3 state: waiting for i/o request (read thread)

I/O thread 4 state: waiting for i/o request (read thread)

I/O thread 5 state: waiting for i/o request (read thread)

I/O thread 6 state: waiting for i/o request (write thread)

I/O thread 7 state: waiting for i/o request (write thread)

I/O thread 8 state: waiting for i/o request (write thread)

I/O thread 9 state: waiting for i/o request (write thread)

Pending normal aio reads: [0, 0, 0, 0] , aio writes: [0, 0, 0, 0] ,

ibuf aio reads:, log i/o's:, sync i/o's:

Pending flushes (fsync) log: 0; buffer pool: 0