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` 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` 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` option enabled.

  The default setting for `innodb_stats_on_metadata` is `OFF`. Enabling `innodb_stats_on_metadata` 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` is configured globally using a `SET` statement.

  ```
  SET GLOBAL innodb_stats_on_metadata=ON
  ```

  

  Note

  `innodb_stats_on_metadata` only applies when optimizer statistics are configured to be non-persistent (when `innodb_stats_persistent` 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`. 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` is a global parameter that can be set at runtime.

The value of `innodb_stats_transient_sample_pages` affects the index sampling for all `InnoDB` tables and indexes when `innodb_stats_persistent=0`. 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` 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` 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` 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`.
- 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` * 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` 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`, the number of pages sampled is defined by `innodb_stats_transient_sample_pages`. See Section 15.8.10.2, "Configuring Non-Persistent Optimizer Statistics Parameters" for additional information.

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

In 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`)
- `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    | n_diff_pfx01 | a                |
  | PRIMARY    | n_diff_pfx02 | a,b              |
  | i1uniq     | n_diff_pfx01 | c                |
  | i1uniq     | n_diff_pfx02 | c,d              |
```

```
  | i2nonuniq  | n_diff_pfx01 | e                |
  | i2nonuniq  | n_diff_pfx02 | e,f              |
  | i2nonuniq  | n_diff_pfx03 | e,f,a            |
  | i2nonuniq  | n_diff_pfx04 | e,f,a,b          |
  | i3nonuniq  | n_diff_pfx01 | g                |
  | i3nonuniq  | n_diff_pfx02 | g,h              |
  | i3nonuniq  | n_diff_pfx03 | g,h,a            |
  | i3nonuniq  | n_diff_pfx04 | g,h,a,b          |
  +------------+--------------+------------------+
```

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

- `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`=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:

```
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
       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 |
+-----------------------------+---------------------------------------+
```

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` and `innodb_log_files_in_group`.

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`

Only consider enabling `innodb_dedicated_server` if the MySQL instance resides on a dedicated server where it can use all available system resources. For example, consider enabling `innodb_dedicated_server` if you run MySQL Server in a Docker container or dedicated VM that only runs MySQL. Enabling `innodb_dedicated_server` 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`

  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`

  Redo log capacity is configured according to the amount of memory detected on the server and, in some cases, whether `innodb_buffer_pool_size` is configured explicitly. If `innodb_buffer_pool_size` is not configured explicitly, the default value is assumed.

Warning

Automatic redo log capacity configuration behavior is undefined if `innodb_buffer_pool_size` 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` (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` (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` 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` value of 2 is enforced if the rounded buffer pool size value is less than 2GB.

- `innodb_flush_method`

  The flush method is set to `O_DIRECT_NO_FSYNC` when `innodb_dedicated_server` is enabled. If the `O_DIRECT_NO_FSYNC` setting is not available, the default `innodb_flush_method` 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` is enabled and `innodb_buffer_pool_size` 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` 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` 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` 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` 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` 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` 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` 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` value. If you specify a value greater than the `innodb_page_size` value, the specified value is ignored, a warning is issued, and `KEY_BLOCK_SIZE` is set to half of the `innodb_page_size` 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` 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”. 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 InnoDB Tables”.

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` is enabled (the default), `CREATE TEMPORARY TABLE` returns errors if `ROW_FORMAT=COMPRESSED` or `KEY_BLOCK_SIZE` is specified. If `innodb_strict_mode` 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 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” 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. 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` 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`, `innodb_compression_failure_threshold_pct`, and `innodb_compression_pad_pct_max` options as described in Section 15.9.1.6, “Compression for OLTP Workloads”, 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 `BLOB`s 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 `INSERT`s 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.

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”, 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 Compression Information Schema Tables”. 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` 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, 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` 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`, and `innodb_compression_pad_pct_max` 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` 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` 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` 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`. 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` 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` must be enabled. There is no dependence on the `innodb_file_per_table` 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` 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` 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`
- `innodb_compression_pad_pct_max` 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` 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` 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` 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` 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` 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` 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` 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` 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` 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” 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” 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 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 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

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 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
       KEY_BLOCK_SIZE=8;
ERROR 1478 (HY000): InnoDB: Tablespace `ts1` cannot contain a COMPRESSED table
```

- 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
       KEY_BLOCK_SIZE=4;
ERROR 1478 (HY000): InnoDB: Tablespace `ts2` uses block size 8192 and cannot
contain a table with physical page size 4096
```

  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` is not applicable to general tablespaces. Tablespace management rules for general tablespaces are strictly enforced independently of `innodb_strict_mode`. 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 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 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 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 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:

- `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      INT             NOT NULL,
    birth_date  DATE            NOT NULL,
    first_name  VARCHAR(14)     NOT NULL,
    last_name   VARCHAR(16)     NOT NULL,
    gender      ENUM ('M','F')  NOT NULL,
    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
       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
```

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
       WHERE CREATE_OPTIONS LIKE '%COMPRESSION=%';
+------------+--------------+--------------------+
| TABLE_NAME | TABLE_SCHEMA | CREATE_OPTIONS     |
+------------+--------------+--------------------+
| employees  | test         | COMPRESSION="zlib" |
+------------+--------------+--------------------+
```

`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`.
- 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
- COMPACT Row Format
- DYNAMIC Row Format
- COMPRESSED Row Format
- Defining the Row Format of a Table
- Determining the Row Format of a Table

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 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 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.

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 Table and Page Compression”.

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 variable must be enabled. The innodb_file_per_table 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.

Defining the Row Format of a Table

The default row format for InnoDB tables is defined by innodb_default_row_format 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 variable can be set dynamically:

```
mysql> SET GLOBAL innodb_default_row_format=DYNAMIC;
```

Valid innodb_default_row_format 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` 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` 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` 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` 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 Operations” 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` 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` 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). 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 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 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` 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` 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` 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` default setting is 12.5 percent, which is the same percentage of pages reserved in previous MySQL releases.

The `innodb_segment_reserve_factor` 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` 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` 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”.

- *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 Row Formats”.

`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 DDL. For more information, see Section 15.12, “InnoDB and Online DDL”.

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` 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.
- 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
- Primary Key Operations
- Column Operations
- Generated Column Operations
- Foreign Key Operations
- Table Operations
- Tablespace Operations
- Partitioning Operations

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 Notes.

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”.

- 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.

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 Notes.

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 c1 VARCHAR(256);
ERROR 0A000: ALGORITHM=INPLACE is not supported. Reason: Cannot change
column type INPLACE. Try ALGORITHM=COPY.
```

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.

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 Notes.

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

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.

Table Operations

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 Notes.

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_SIZE`	No	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, L
```

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.

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”.

- 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”.

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.

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*	`ALGORITHM=INPLACE, LOCK={DEFAULT|NONE|SHARED|EXCLUSISVE}` is supported for `RANGE` and `LIST` partitions, `ALGORITHM=INPLACE, 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*	`ALGORITHM=INPLACE, LOCK={DEFAULT|NONE|SHARED|EXCLUSIVE}` is supported. Does not copy data for tables partitioned by `RANGE` or `LIST`. `DROP PARTITION` with `ALGORITHM=INPLACE` deletes data stored in the partition and drops the partition. However, `DROP PARTITION` with `ALGORITHM=COPY`

Partitioning Clause	Instant	In Place	Permits DML	Notes
				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.
`DISCARD PARTITION`	No	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`
`IMPORT PARTITION`	No	No	No	Only permits `ALGORITHM=DEFAULT`, `LOCK=DEFAULT`
`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	`ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSIVE}` is supported.
`REORGANIZE PARTITION`	No	Yes*	No	`ALGORITHM=INPLACE, LOCK={DEFAULT|SHARED|EXCLUSIVE}` is supported.
`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,

Partitioning Clause	Instant	In Place	Permits DML	Notes
				"Maintenance of Partitions".
`REBUILD PARTITION`	No	Yes*	No	`ALGORITHM=INPLACE, 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 Tables Using Performance Schema”.

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` up to a maximum specified by `innodb_online_alter_log_max_size`. 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`, 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` 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` 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`. 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` 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` 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`/`innodb_ddl_threads`).

Prior to MySQL 8.0.27, `innodb_sort_buffer_size` 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` 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` 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` 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` 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 Space Requirements”.
- 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` 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”.

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

`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 Tablespaces.

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 and General Tablespaces.

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` 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` 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` 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` 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` 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` 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.

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 Prerequisites.)

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.

```
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
       WHERE CREATE_OPTIONS LIKE '%ENCRYPTION%';
+--------------+------------+----------------+
| TABLE_SCHEMA | TABLE_NAME | CREATE_OPTIONS |
+--------------+------------+----------------+
| test         | t1         | ENCRYPTION="Y" |
+--------------+------------+----------------+
```

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
       WHERE DEFAULT_ENCRYPTION='YES';
+-------------+--------------------+
| SCHEMA_NAME | DEFAULT_ENCRYPTION |
+-------------+--------------------+
| test        | YES                |
+-------------+--------------------+
```

`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:

```
mysql> USE performance_schema;
mysql> UPDATE setup_instruments SET ENABLED = 'YES'
       WHERE NAME LIKE 'stage/innodb/alter tablespace (encryption)';
```

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` and `innodb_undo_log_encrypt`. See Redo Log Encryption, and Undo Log Encryption. 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` or `--skip-innodb-adaptive-hash-index` on the command line, or `innodb_adaptive_hash_index` or `skip_innodb_adaptive_hash_index` in an option file.
- Some variable descriptions refer to “enabling” or “disabling” a variable. These variables can be enabled with the `SET` statement by setting them to `ON` or `1`, or disabled by setting them to `OFF` or `0`. Boolean variables can be set at startup to the values `ON`, `TRUE`, `OFF`, and `FALSE` (not case-sensitive), as well as `1` and `0`. See Section 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_memcached_enable_binlog	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_name	Yes	Yes	Yes		Global	No
daemon_memcached_engine_lib_path	Yes	Yes	Yes		Global	No
daemon_memcached_option	Yes	Yes	Yes		Global	No
daemon_memcached_r_batch_size	Yes	Yes	Yes		Global	No
daemon_memcached_w_batch_size	Yes	Yes	Yes		Global	No
foreign_key_checks			Yes		Both	Yes
innodb	Yes	Yes				
innodb_adaptive_flushing	Yes	Yes	Yes		Global	Yes
innodb_adaptive_flushing_lwm	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index	Yes	Yes	Yes		Global	Yes
innodb_adaptive_hash_index_parts	Yes	Yes	Yes		Global	No
innodb_adaptive_max_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_api_bk_commit_interval	Yes	Yes	Yes		Global	Yes
innodb_api_disable_rowlock	Yes	Yes	Yes		Global	No
innodb_api_enable_binlog	Yes	Yes	Yes		Global	No
innodb_api_enable_mdl	Yes	Yes	Yes		Global	No
innodb_api_trx_level	Yes	Yes	Yes		Global	Yes
innodb_autoextend_increment	Yes	Yes	Yes		Global	Yes
innodb_autoinc_lock_mode	Yes	Yes	Yes		Global	No
innodb_background_drop_list_empty	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_bytes_data				Yes	Global	No
Innodb_buffer_pool_bytes_dirty				Yes	Global	No
innodb_buffer_pool_chunk_size	Yes	Yes	Yes		Global	No
innodb_buffer_pool_debug	Yes	Yes	Yes		Global	No
innodb_buffer_pool_dump_at_shutdown	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_now	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_dump_pct	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_dump_status				Yes	Global	No
innodb_buffer_pool_filename	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_in_core_file	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_instances	Yes	Yes	Yes		Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
innodb_buffer_pool_load_abort	Yes	Yes	Yes		Global	Yes
innodb_buffer_pool_load_at_startup	Yes	Yes	Yes		Global	No
innodb_buffer_pool_load_now	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_load_status				Yes	Global	No
Innodb_buffer_pool_pages_data				Yes	Global	No
Innodb_buffer_pool_pages_dirty				Yes	Global	No
Innodb_buffer_pool_pages_flushed				Yes	Global	No
Innodb_buffer_pool_pages_free				Yes	Global	No
Innodb_buffer_pool_pages_latched				Yes	Global	No
Innodb_buffer_pool_pages_misc				Yes	Global	No
Innodb_buffer_pool_pages_total				Yes	Global	No
Innodb_buffer_pool_read_ahead				Yes	Global	No
Innodb_buffer_pool_read_ahead_evicted				Yes	Global	No
Innodb_buffer_pool_read_ahead_rnd				Yes	Global	No
Innodb_buffer_pool_read_requests				Yes	Global	No
Innodb_buffer_pool_reads				Yes	Global	No
Innodb_buffer_pool_resize_status				Yes	Global	No
innodb_buffer_pool_size	Yes	Yes	Yes		Global	Yes
Innodb_buffer_pool_wait_free				Yes	Global	No
Innodb_buffer_pool_write_requests				Yes	Global	No
innodb_change_buffer_max_size	Yes	Yes	Yes		Global	Yes
innodb_change_buffering	Yes	Yes	Yes		Global	Yes
innodb_change_buffering_debug	Yes	Yes	Yes		Global	Yes
innodb_checkpoint_disabled	Yes	Yes	Yes		Global	Yes
innodb_checksum_algorithm	Yes	Yes	Yes		Global	Yes
innodb_cmp_per_index_enabled	Yes	Yes	Yes		Global	Yes
innodb_commit_concurrency	Yes	Yes	Yes		Global	Yes
innodb_compress_debug	Yes	Yes	Yes		Global	Yes
innodb_compression_failure_threshold_pct	Yes	Yes	Yes		Global	Yes
innodb_compression_level	Yes	Yes	Yes		Global	Yes
innodb_compression_pad_pct_max	Yes	Yes	Yes		Global	Yes
innodb_concurrency_tickets	Yes	Yes	Yes		Global	Yes
innodb_data_file_path	Yes	Yes	Yes		Global	No
Innodb_data_fsyncs				Yes	Global	No
innodb_data_home_dir	Yes	Yes	Yes		Global	No
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_buffer_size	Yes	Yes	Yes		Both	Yes
innodb_ddl_log_crash_reset_debug	Yes	Yes	Yes		Global	Yes
innodb_ddl_threads	Yes	Yes	Yes		Both	Yes
innodb_deadlock_detect	Yes	Yes	Yes		Global	Yes
innodb_dedicated_server	Yes	Yes	Yes		Global	No
innodb_default_row_format	Yes	Yes	Yes		Global	Yes
innodb_directories	Yes	Yes	Yes		Global	No
innodb_disable_sort_file_cache	Yes	Yes	Yes		Global	Yes
innodb_doublewrite	Yes	Yes	Yes		Global	Varies
innodb_doublewrite_batch_size	Yes	Yes	Yes		Global	No
innodb_doublewrite_dir	Yes	Yes	Yes		Global	No
innodb_doublewrite_files	Yes	Yes	Yes		Global	No
innodb_doublewrite_pages	Yes	Yes	Yes		Global	No
innodb_fast_shutdown	Yes	Yes	Yes		Global	Yes
innodb_fil_make_page_dirty_debug	Yes	Yes	Yes		Global	Yes
innodb_file_per_table	Yes	Yes	Yes		Global	Yes
innodb_fill_factor	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_timeout	Yes	Yes	Yes		Global	Yes
innodb_flush_log_at_trx_commit	Yes	Yes	Yes		Global	Yes
innodb_flush_method	Yes	Yes	Yes		Global	No
innodb_flush_neighbors	Yes	Yes	Yes		Global	Yes
innodb_flush_sync	Yes	Yes	Yes		Global	Yes
innodb_flushing_avg_loops	Yes	Yes	Yes		Global	Yes
innodb_force_load_corrupted	Yes	Yes	Yes		Global	No
innodb_force_recovery	Yes	Yes	Yes		Global	No
innodb_fsync_threshold	Yes	Yes	Yes		Global	Yes
innodb_ft_aux_table			Yes		Global	Yes
innodb_ft_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_enable_diag_print	Yes	Yes	Yes		Global	Yes
innodb_ft_enable_stopword	Yes	Yes	Yes		Both	Yes
innodb_ft_max_token_size	Yes	Yes	Yes		Global	No
innodb_ft_min_token_size	Yes	Yes	Yes		Global	No
innodb_ft_num_word_optimize	Yes	Yes	Yes		Global	Yes
innodb_ft_result_cache_limit	Yes	Yes	Yes		Global	Yes
innodb_ft_server_stopword_table	Yes	Yes	Yes		Global	Yes
innodb_ft_sort_pll_degree	Yes	Yes	Yes		Global	No
innodb_ft_total_cache_size	Yes	Yes	Yes		Global	No
innodb_ft_user_stopword_table	Yes	Yes	Yes		Both	Yes

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Innodb_have_atomic_builtins				Yes	Global	No
innodb_idle_flush_pct	Yes	Yes	Yes		Global	Yes
innodb_io_capacity	Yes	Yes	Yes		Global	Yes
innodb_io_capacity_max	Yes	Yes	Yes		Global	Yes
innodb_limit_optimistic_insert_debug	Yes	Yes	Yes		Global	Yes
innodb_lock_wait_timeout	Yes	Yes	Yes		Both	Yes
innodb_log_buffer_size	Yes	Yes	Yes		Global	Varies
innodb_log_checkpoint_fuzzy_now	Yes	Yes	Yes		Global	Yes
innodb_log_checkpoint_now	Yes	Yes	Yes		Global	Yes
innodb_log_checksums	Yes	Yes	Yes		Global	Yes
innodb_log_compressed_pages	Yes	Yes	Yes		Global	Yes
innodb_log_file_size	Yes	Yes	Yes		Global	No
innodb_log_files_in_group	Yes	Yes	Yes		Global	No
innodb_log_group_home_dir	Yes	Yes	Yes		Global	No
innodb_log_spin_cpu_abs_lwm	Yes	Yes	Yes		Global	Yes
innodb_log_spin_cpu_pct_hwm	Yes	Yes	Yes		Global	Yes
innodb_log_wait_for_flush_spin_hwm	Yes	Yes	Yes		Global	Yes
Innodb_log_waits				Yes	Global	No
innodb_log_write_ahead_size	Yes	Yes	Yes		Global	Yes
Innodb_log_write_requests				Yes	Global	No
innodb_log_writer_threads	Yes	Yes	Yes		Global	Yes
Innodb_log_writes				Yes	Global	No
innodb_lru_scan_depth	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct	Yes	Yes	Yes		Global	Yes
innodb_max_dirty_pages_pct_lwm	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag	Yes	Yes	Yes		Global	Yes
innodb_max_purge_lag_delay	Yes	Yes	Yes		Global	Yes
innodb_max_undo_log_size	Yes	Yes	Yes		Global	Yes
innodb_merge_threshold_set_all_debug	Yes	Yes	Yes		Global	Yes
innodb_monitor_disable	Yes	Yes	Yes		Global	Yes
innodb_monitor_enable	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset	Yes	Yes	Yes		Global	Yes
innodb_monitor_reset_all	Yes	Yes	Yes		Global	Yes
Innodb_num_open_files				Yes	Global	No
innodb_numa_interleave	Yes	Yes	Yes		Global	No
innodb_old_blocks_pct	Yes	Yes	Yes		Global	Yes
innodb_old_blocks_time	Yes	Yes	Yes		Global	Yes
innodb_online_alter_log_max_size	Yes	Yes	Yes		Global	Yes
innodb_open_files	Yes	Yes	Yes		Global	Varies
innodb_optimize_fulltext_only	Yes	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_fsyncs				Yes	Global	No
Innodb_os_log_pending_writes				Yes	Global	No
Innodb_os_log_written				Yes	Global	No
innodb_page_cleaners	Yes	Yes	Yes		Global	No
Innodb_page_size				Yes	Global	No
innodb_page_size	Yes	Yes	Yes		Global	No
Innodb_pages_created				Yes	Global	No
Innodb_pages_read				Yes	Global	No
Innodb_pages_written				Yes	Global	No
innodb_parallel_read_threads	Yes	Yes	Yes		Session	Yes
innodb_print_all_deadlocks	Yes	Yes	Yes		Global	Yes
innodb_print_ddl_logs	Yes	Yes	Yes		Global	Yes
innodb_purge_batch_size	Yes	Yes	Yes		Global	Yes
innodb_purge_rseg_truncate_frequency	Yes	Yes	Yes		Global	Yes
innodb_purge_threads	Yes	Yes	Yes		Global	No
innodb_random_read_ahead	Yes	Yes	Yes		Global	Yes
innodb_read_ahead_threshold	Yes	Yes	Yes		Global	Yes
innodb_read_io_threads	Yes	Yes	Yes		Global	No
innodb_read_only	Yes	Yes	Yes		Global	No
innodb_redo_log_archive_dirs	Yes	Yes	Yes		Global	Yes
innodb_redo_log_capacity	Yes	Yes	Yes		Global	Yes
Innodb_redo_log_capacity_resized				Yes	Global	No
Innodb_redo_log_checkpoint_lsn				Yes	Global	No
Innodb_redo_log_current_lsn				Yes	Global	No
Innodb_redo_log_enabled				Yes	Global	No
innodb_redo_log_encrypt	Yes	Yes	Yes		Global	Yes
Innodb_redo_log_flushed_to_disk_lsn				Yes	Global	No
Innodb_redo_log_logical_size				Yes	Global	No
Innodb_redo_log_physical_size				Yes	Global	No
Innodb_redo_log_read_only				Yes	Global	No
Innodb_redo_log_resize_status				Yes	Global	No
Innodb_redo_log_uuid				Yes	Global	No
innodb_replication_delay	Yes	Yes	Yes		Global	Yes
innodb_rollback_on_timeout	Yes	Yes	Yes		Global	No
innodb_rollback_segments	Yes	Yes	Yes		Global	Yes
Innodb_row_lock_current_waits				Yes	Global	No
Innodb_row_lock_time				Yes	Global	No
Innodb_row_lock_time_avg				Yes	Global	No
Innodb_row_lock_time_max				Yes	Global	No
Innodb_row_lock_waits				Yes	Global	No
Innodb_rows_deleted				Yes	Global	No

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_page_number_debug	Yes	Yes	Yes		Global	Yes
innodb_segment_reserve_factor	Yes	Yes	Yes		Global	Yes
innodb_sort_buffer_size	Yes	Yes	Yes		Global	No
innodb_spin_wait_delay	Yes	Yes	Yes		Global	Yes
innodb_spin_wait_pause_multiplier	Yes	Yes	Yes		Global	Yes
innodb_stats_auto_recalc	Yes	Yes	Yes		Global	Yes
innodb_stats_include_delete_marked	Yes	Yes	Yes		Global	Yes
innodb_stats_method	Yes	Yes	Yes		Global	Yes
innodb_stats_on_metadata	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent	Yes	Yes	Yes		Global	Yes
innodb_stats_persistent_sample_pages	Yes	Yes	Yes		Global	Yes
innodb_stats_transient_sample_pages	Yes	Yes	Yes		Global	Yes
innodb-status-file	Yes	Yes				
innodb_status_output	Yes	Yes	Yes		Global	Yes
innodb_status_output_locks	Yes	Yes	Yes		Global	Yes
innodb_strict_mode	Yes	Yes	Yes		Both	Yes
innodb_sync_array_size	Yes	Yes	Yes		Global	No
innodb_sync_debug	Yes	Yes	Yes		Global	No
innodb_sync_spin_loops	Yes	Yes	Yes		Global	Yes
Innodb_system_rows_deleted				Yes	Global	No
Innodb_system_rows_inserted				Yes	Global	No
Innodb_system_rows_read				Yes	Global	No
innodb_table_locks	Yes	Yes	Yes		Both	Yes
innodb_temp_data_file_path	Yes	Yes	Yes		Global	No
innodb_temp_tablespaces_dir	Yes	Yes	Yes		Global	No
innodb_thread_concurrency	Yes	Yes	Yes		Global	Yes
innodb_thread_sleep_delay	Yes	Yes	Yes		Global	Yes
innodb_tmpdir	Yes	Yes	Yes		Both	Yes
Innodb_truncated_status_writes				Yes	Global	No
innodb_trx_purge_view_update_only_debug	Yes	Yes	Yes		Global	Yes
innodb_trx_rseg_n_slots_debug	Yes	Yes	Yes		Global	Yes
innodb_undo_directory	Yes	Yes	Yes		Global	No
innodb_undo_log_encrypt	Yes	Yes	Yes		Global	Yes
innodb_undo_log_truncate	Yes	Yes	Yes		Global	Yes
innodb_undo_tablespaces	Yes	Yes	Yes		Global	Varies
Innodb_undo_tablespaces_active				Yes	Global	No
Innodb_undo_tablespaces_explicit				Yes	Global	No

Name	Cmd-Line	Option File	System Var	Status Var	Var Scope	Dynamic
Innodb_undo_tablespaces_implicit				Yes	Global	No
Innodb_undo_tablespaces_total				Yes	Global	No
innodb_use_fdatasync	Yes	Yes	Yes		Global	Yes
innodb_use_native_aio	Yes	Yes	Yes		Global	No
innodb_validate_tablespace_paths	Yes	Yes	Yes		Global	No
innodb_version			Yes		Global	No
innodb_write_io_threads	Yes	Yes	Yes		Global	No
unique_checks			Yes		Both	Yes

InnoDB Command Options

- `--innodb[=value]`

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

Valid Values	`OFF`
	`ON`
	`FORCE`

Controls loading of the `InnoDB` storage engine, if the server was compiled with `InnoDB` support. This option has a tristate format, with possible values of `OFF`, `ON`, or `FORCE`. See Section 5.6.1, “Installing and Uninstalling Plugins”.

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

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

- `--innodb-status-file`

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

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

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

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

For related information, see Section 15.17.2, “Enabling InnoDB Monitors”.

- `--skip-innodb`

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

InnoDB System Variables

- `daemon_memcached_enable_binlog`

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

Command-Line Format	`--daemon-memcached-engine-lib-name=file_name`
System Variable	`daemon_memcached_engine_lib_name`
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`

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

Command-Line Format	`--daemon-memcached-option=options`
System Variable	`daemon_memcached_option`
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`

Command-Line Format	`--daemon-memcached-r-batch-size=#`
System Variable	`daemon_memcached_r_batch_size`
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`.

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`

Command-Line Format	`--daemon-memcached-w-batch-size=#`
System Variable	`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.

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

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

Specifies whether to dynamically adjust the rate of flushing dirty pages in the InnoDB buffer pool based on the workload. Adjusting the flush rate dynamically is intended to avoid bursts of I/O activity. This setting is enabled by default. See Section 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

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

Defines the low water mark representing percentage of redo log capacity at which adaptive flushing is enabled. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_adaptive_hash_index`

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

Command-Line Format	`--innodb-adaptive-hash-index-parts=#`
System Variable	`innodb_adaptive_hash_index_parts`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Numeric
Default Value	`8`
Minimum Value	`1`
Maximum Value	`512`

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

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

For related information, see Section 15.5.3, “Adaptive Hash Index”.

- `innodb_adaptive_max_sleep_delay`

Command-Line Format	`--innodb-adaptive-max-sleep-delay=#`
System Variable	`innodb_adaptive_max_sleep_delay`
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	150000
Minimum Value	0
Maximum Value	1000000
Unit	microseconds

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

For more information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- innodb_api_bk_commit_interval

Command-Line Format	--innodb-api-bk-commit-interval=#
System Variable	innodb_api_bk_commit_interval
Scope	Global
Dynamic	Yes
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

Command-Line Format	--innodb-api-disable-rowlock[={OFF|ON}]
System Variable	innodb_api_disable_rowlock
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 is disabled, which means that memcached requests row locks for

`get` and `set` operations. When `innodb_api_disable_rowlock` is enabled, `memcached` requests a table lock instead of row locks.

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

For more information, see Section 15.20.6.4, "Controlling Transactional Behavior of the InnoDB memcached Plugin".

- `innodb_api_enable_binlog`

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

Command-Line Format	`--innodb-api-enable-mdl[={OFF|ON}]`
System Variable	`innodb_api_enable_mdl`
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`

Command-Line Format	`--innodb-api-trx-level=#`
System Variable	`innodb_api_trx_level`
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

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

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

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

- innodb_autoinc_lock_mode

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

Command-Line Format	`--innodb-background-drop-list-empty[={OFF|ON}]`
System Variable	`innodb_background_drop_list_empty`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

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

- `innodb_buffer_pool_chunk_size`

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

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

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

`innodb_buffer_pool_chunk_size` can be increased or decreased in units of 1MB (1048576 bytes).

The following conditions apply when altering the `innodb_buffer_pool_chunk_size` value:

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

Important

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

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

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

See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

- `innodb_buffer_pool_debug`

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

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

- `innodb_buffer_pool_dump_at_shutdown`

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

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

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

Both innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_dump_now

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

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

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

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

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_dump_pct

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

Maximum Value	100

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

- `innodb_buffer_pool_filename`

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

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

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

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

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

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- `innodb_buffer_pool_in_core_file`

Command-Line Format	`--innodb-buffer-pool-in-core-file[={OFF|ON}]`
Introduced	8.0.14
System Variable	`innodb_buffer_pool_in_core_file`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

Disabling the `innodb_buffer_pool_in_core_file` variable reduces the size of core files by excluding `InnoDB` buffer pool pages. To use this variable, the `core_file` variable must be enabled and 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`

Command-Line Format	`--innodb-buffer-pool-instances=#`
System Variable	`innodb_buffer_pool_instances`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value (Windows, 32-bit platforms)	`(autosized)`
Default Value (Other)	`8 (or 1 if innodb_buffer_pool_size < 1GB)`
Minimum Value	`1`
Maximum Value	`64`

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

This option only takes effect when setting `innodb_buffer_pool_size` to 1GB or more. The total buffer pool size is divided among all the buffer pools. For best efficiency, specify a combination of `innodb_buffer_pool_instances` and `innodb_buffer_pool_size` so that each buffer pool instance is at least 1GB.

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

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

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

For related information, see Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size”.

- `innodb_buffer_pool_load_abort`

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

Default Value	OFF

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

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

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_load_at_startup

Command-Line Format	--innodb-buffer-pool-load-at-startup[={OFF|ON}]
System Variable	innodb_buffer_pool_load_at_startup
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

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

Both innodb_buffer_pool_dump_at_shutdown and innodb_buffer_pool_load_at_startup are enabled by default.

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_load_now

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

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

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

For more information, see Section 15.8.3.6, “Saving and Restoring the Buffer Pool State”.

- innodb_buffer_pool_size

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

System Variable	innodb_buffer_pool_size
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	134217728
Minimum Value	5242880
Maximum Value (64-bit platforms)	2**64-1
Maximum Value (32-bit platforms)	2**32-1
Unit	bytes

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

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

- Competition for physical memory can cause paging in the operating system.
- InnoDB reserves additional memory for buffers and control structures, so that the total allocated space is approximately 10% greater than the specified buffer pool size.
- Address space for the buffer pool must be contiguous, which can be an issue on Windows systems with DLLs that load at specific addresses.
- The time to initialize the buffer pool is roughly proportional to its size. On instances with large buffer pools, initialization time might be significant. To reduce the initialization period, you can save the buffer pool state at server shutdown and restore it at server startup. See Section 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 variable, which has a default of 128 MB.

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

innodb_buffer_pool_size can be set dynamically, which allows you to resize the buffer pool without restarting the server. The Innodb_buffer_pool_resize_status status variable reports the status of online buffer pool resizing operations. See Section 15.8.3.1, “Configuring InnoDB Buffer Pool Size” for more information.

If innodb_dedicated_server is enabled, the innodb_buffer_pool_size value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_change_buffer_max_size`

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

Maximum size for the `InnoDB` change buffer, as a percentage of the total size of the buffer pool. You might increase this value for a MySQL server with heavy insert, update, and delete activity, or decrease it for a MySQL server with unchanging data used for reporting. For more information, see Section 15.5.2, “Change Buffer”. For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_change_buffering`

Command-Line Format	`--innodb-change-buffering=value`
System Variable	`innodb_change_buffering`
Scope	Global
Dynamic	Yes
`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

Command-Line Format	--innodb-change-buffering-debug=#
System Variable	innodb_change_buffering_debug
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	2

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

- innodb_checkpoint_disabled

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

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

- innodb_checksum_algorithm

Command-Line Format	--innodb-checksum-algorithm=value

System Variable	innodb_checksum_algorithm
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	crc32
Valid Values	crc32 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 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 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`

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

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

For more information, see Section 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”.

- `innodb_commit_concurrency`

Command-Line Format	`--innodb-commit-concurrency=#`
System Variable	`innodb_commit_concurrency`
Scope	Global
Dynamic	Yes

SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	1000

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

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

- innodb_compress_debug

Command-Line Format	--innodb-compress-debug=value
System Variable	innodb_compress_debug
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Enumeration
Default Value	none
Valid Values	none 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”.

- innodb_compression_failure_threshold_pct

Command-Line Format	--innodb-compression-failure-threshold-pct=#
System Variable	innodb_compression_failure_threshold_pct
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	5
Minimum Value	0
Maximum Value	100

Defines the compression failure rate threshold for a table, as a percentage, at which point MySQL begins adding padding within compressed pages to avoid expensive compression failures. When this threshold is passed, MySQL begins to leave additional free space within each new compressed page, dynamically adjusting the amount of free space up to the percentage of page size specified

by innodb_compression_pad_pct_max. A value of zero disables the mechanism that monitors compression efficiency and dynamically adjusts the padding amount.

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_compression_level

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

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

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_compression_pad_pct_max

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

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

For more information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_concurrency_tickets

Command-Line Format	--innodb-concurrency-tickets=#
System Variable	innodb_concurrency_tickets
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No

Type	Integer
Default Value	5000
Minimum Value	1
Maximum Value	4294967295

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

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

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

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

For more information, see Section 15.8.4, "Configuring Thread Concurrency for InnoDB".

- innodb_data_file_path

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

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

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

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

File sizes are specified in kilobytes, megabytes, or gigabytes by appending K, M or G to the size value. If specifying the data file size in kilobytes, do so in multiples of 1024. Otherwise, KB values are

rounded to nearest megabyte (MB) boundary. The sum of file sizes must be, at a minimum, slightly larger than 12MB.

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

- `innodb_data_home_dir`

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

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

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

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

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

For related information, see Section 15.8.1, “InnoDB Startup Configuration”.

- `innodb_ddl_buffer_size`

Command-Line Format	`--innodb-ddl-buffer-size=#`
Introduced	8.0.27
System Variable	`innodb_ddl_buffer_size`
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”. The maximum buffer size per DDL thread is the maximum buffer size divided by the number of DDL threads (`innodb_ddl_buffer_size`/`innodb_ddl_threads`).

- `innodb_ddl_log_crash_reset_debug`

Command-Line Format	`--innodb-ddl-log-crash-reset-debug[={OFF|ON}]`
System Variable	`innodb_ddl_log_crash_reset_debug`

Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

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

- innodb_ddl_threads

Command-Line Format	--innodb-ddl-threads=#
Introduced	8.0.27
System Variable	innodb_ddl_threads
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”, and Section 15.12.4, “Online DDL Memory Management”.

- innodb_deadlock_detect

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

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

For related information, see Section 15.7.5.2, “Deadlock Detection”.

- innodb_dedicated_server

Command-Line Format	--innodb-dedicated-server[={OFF|ON}]
System Variable	innodb_dedicated_server
Scope	Global
Dynamic	No

SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

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

- innodb_buffer_pool_size
- innodb_redo_log_capacity or, prior to MySQL 8.0.30, innodb_log_file_size and innodb_log_files_in_group.

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

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

For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- innodb_default_row_format

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

The innodb_default_row_format option defines the default row format for InnoDB tables and user-created temporary tables. The default setting is DYNAMIC. Other permitted values are

COMPACT and REDUNDANT. The COMPRESSED row format, which is not supported for use in the system tablespace, cannot be defined as the default.

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

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

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

- innodb_directories

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

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

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

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

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

Startup command:

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

MySQL option file:

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

Wildcard expressions cannot be used to specify directories.

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

For more information, see Section 15.6.3.6, “Moving Tablespace Files While the Server is Offline”.

- `innodb_disable_sort_file_cache`

Command-Line Format	`--innodb-disable-sort-file-cache[={OFF|ON}]`
System Variable	`innodb_disable_sort_file_cache`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`OFF`

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

- `innodb_doublewrite`

Command-Line Format	`--innodb-doublewrite=value` (≥ 8.0.30) `--innodb-doublewrite[={OFF|ON}]` (≤ 8.0.29)
System Variable	`innodb_doublewrite`
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` variable controls doublewrite buffering. Doublewrite buffering is enabled by default in most cases.

Prior to MySQL 8.0.30, you can set `innodb_doublewrite` to `ON` or `OFF` when starting the server to enable or disable doublewrite buffering, respectively. From MySQL 8.0.30, `innodb_doublewrite` 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` setting that enables the doublewrite buffer, between `ON`, `DETECT_AND_RECOVER`, and `DETECT_ONLY`. MySQL

does not support dynamic changes between a setting that enables the doublewrite buffer and `OFF` or vice versa.

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

For related information, see Section 15.6.4, “Doublewrite Buffer”.

- `innodb_doublewrite_batch_size`

Command-Line Format	`--innodb-doublewrite-batch-size=#`
Introduced	8.0.20
System Variable	`innodb_doublewrite_batch_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`256`

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

For more information, see Section 15.6.4, “Doublewrite Buffer”.

- `innodb_doublewrite_dir`

Command-Line Format	`--innodb-doublewrite-dir=dir_name`
Introduced	8.0.20
System Variable	`innodb_doublewrite_dir`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Directory name

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

For more information, see Section 15.6.4, “Doublewrite Buffer”.

- `innodb_doublewrite_files`

Command-Line Format	`--innodb-doublewrite-files=#`
Introduced	8.0.20
System Variable	`innodb_doublewrite_files`
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 variable.)

For more information, see Section 15.6.4, “Doublewrite Buffer”.

- innodb_doublewrite_pages

Command-Line Format	--innodb-doublewrite-pages=#
Introduced	8.0.20
System Variable	innodb_doublewrite_pages
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 is set to the innodb_write_io_threads value.

For more information, see Section 15.6.4, “Doublewrite Buffer”.

- innodb_extend_and_initialize

Command-Line Format	--innodb=extend-and-initialize[={OFF|ON}]
Introduced	8.0.22
System Variable	innodb_extend_and_initialize
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

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

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

For more information, see Section 15.6.3.8, “Optimizing Tablespace Space Allocation on Linux”.

- `innodb_fast_shutdown`

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

Valid Values	`0`
	`1`
	`2`

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

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

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

- `innodb_fil_make_page_dirty_debug`

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

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

- `innodb_file_per_table`

Command-Line Format	`--innodb-file-per-table[={OFF|ON}]`
System Variable	`innodb_file_per_table`
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

When innodb_file_per_table is enabled, tables are created in file-per-table tablespaces by default. When disabled, tables are created in the system tablespace by default. For information about file-per-table tablespaces, see Section 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 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 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

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

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

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

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

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

For more information, see Section 15.6.2.3, “Sorted Index Builds”.

- `innodb_flush_log_at_timeout`

Command-Line Format	`--innodb-flush-log-at-timeout=#`
System Variable	`innodb_flush_log_at_timeout`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1`
Minimum Value	`1`
Maximum Value	`2700`
Unit	seconds

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

- `innodb_flush_log_at_trx_commit`

Command-Line Format	`--innodb-flush-log-at-trx-commit=#`
System Variable	`innodb_flush_log_at_trx_commit`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`1`

Valid Values	`0`
	`1`
	`2`

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

- The default setting of 1 is required for full ACID compliance. Logs are written and flushed to disk at each transaction commit.
- With a setting of 0, logs are written and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- With a setting of 2, logs are written after each transaction commit and flushed to disk once per second. Transactions for which logs have not been flushed can be lost in a crash.
- For settings 0 and 2, once-per-second flushing is not 100% guaranteed. Flushing may occur more frequently due to DDL changes and other internal `InnoDB` activities that cause logs to be flushed independently of the `innodb_flush_log_at_trx_commit` setting, and sometimes less frequently due to scheduling issues. If logs are flushed once per second, up to one second of transactions can be lost in a crash. If logs are flushed more or less frequently than once per second, the amount of transactions that can be lost varies accordingly.

- Log flushing frequency is controlled by `innodb_flush_log_at_timeout`, which allows you to set log flushing frequency to *`N`* seconds (where *`N`* is `1 ... 2700`, with a default value of 1). However, any unexpected `mysqld` process exit can erase up to *`N`* seconds of transactions.
- DDL changes and other internal `InnoDB` activities flush the log independently of the `innodb_flush_log_at_trx_commit` setting.
- `InnoDB` crash recovery works regardless of the `innodb_flush_log_at_trx_commit` setting. Transactions are either applied entirely or erased entirely.

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

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

For information on the combination of settings on a replica that is most resilient to unexpected halts, see Section 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`

Command-Line Format	`--innodb-flush-method=value`
System Variable	`innodb_flush_method`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	String
Default Value (Unix)	`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 options can be specified numerically.

The innodb_flush_method options for Unix-like systems include:

- fsync or 0: InnoDB uses the fsync() system call to flush both the data and log files. 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 variable, introduced in MySQL 8.0.26, permits innodb_flush_method options that use fsync() to use fdatasync() instead. An fdatasync() system call does not flush changes to file metadata unless required for subsequent data retrieval, providing a potential performance benefit.

The innodb_flush_method options for Windows systems include:

- unbuffered or 0: InnoDB uses 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` 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` is enabled, the `innodb_flush_method` value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_flush_neighbors`

Command-Line Format	`--innodb-flush-neighbors=#`
System Variable	`innodb_flush_neighbors`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`0`

Valid Values	`0`
	`1`
	`2`

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

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

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

SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

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

For information about configuring the innodb_flush_sync variable, see Section 15.8.7, “Configuring InnoDB I/O Capacity”.

- innodb_flushing_avg_loops

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

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

For related information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- innodb_force_load_corrupted

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

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

- innodb_force_recovery

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

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

The crash recovery mode, typically only changed in serious troubleshooting situations. Possible values are from 0 to 6. For the meanings of these values and important information about innodb_force_recovery, see Section 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 is greater than 0. An innodb_force_recovery setting of 4 or greater places InnoDB into read-only mode.

These restrictions may cause replication administration commands to fail with an error, as replication stores the replica status logs in InnoDB tables.

- innodb_fsync_threshold

Command-Line Format	--innodb-fsync-threshold=#
Introduced	8.0.13
System Variable	innodb_fsync_threshold
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	0
Minimum Value	0
Maximum Value	2**64-1

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

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

- innodb_ft_aux_table

System Variable	innodb_ft_aux_table

Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	String

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

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

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

For more information, see Section 15.15.4, "InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables".

- innodb_ft_cache_size

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

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

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

- innodb_ft_enable_diag_print

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

Default Value	OFF

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

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

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

- FTS optimize progress. For example:

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

- FTS index build progress. For example:

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

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

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

- `innodb_ft_enable_stopword`

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

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

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- `innodb_ft_max_token_size`

Command-Line Format	`--innodb-ft-max-token-size=#`
System Variable	`innodb_ft_max_token_size`
Scope	Global
Dynamic	No
`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`

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

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

For more information, see Section 12.10.6, “Fine-Tuning MySQL Full-Text Search”.

- `innodb_ft_num_word_optimize`

Command-Line Format	`--innodb-ft-num-word-optimize=#`
System Variable	`innodb_ft_num_word_optimize`
Scope	Global
Dynamic	Yes
`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`

Command-Line Format	`--innodb-ft-result-cache-limit=#`
System Variable	`innodb_ft_result_cache_limit`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No

Type	Integer
Default Value	2000000000
Minimum Value	1000000
Maximum Value	2**32-1
Unit	bytes

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

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

- innodb_ft_server_stopword_table

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

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

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

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

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

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- innodb_ft_sort_pll_degree

Command-Line Format	--innodb-ft-sort-pll-degree=#
System Variable	innodb_ft_sort_pll_degree
Scope	Global
Dynamic	No
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.

- innodb_ft_total_cache_size

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

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

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

- innodb_ft_user_stopword_table

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

Default Value	NULL

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

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

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

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

For more information, see Section 12.10.4, “Full-Text Stopwords”.

- innodb_idle_flush_pct

Command-Line Format	--innodb-idle-flush-pct=#
Introduced	8.0.18
System Variable	innodb_idle_flush_pct
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	100
Minimum Value	0
Maximum Value	100

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

- innodb_io_capacity

Command-Line Format	--innodb-io-capacity=#
System Variable	innodb_io_capacity
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	200
Minimum Value	100
Maximum Value (64-bit platforms)	2**64-1

Maximum Value (32-bit platforms)	`2**32-1`

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

For information about configuring the `innodb_io_capacity` variable, see Section 15.8.7, "Configuring InnoDB I/O Capacity".

- `innodb_io_capacity_max`

Command-Line Format	`--innodb-io-capacity-max=#`
System Variable	`innodb_io_capacity_max`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`see description`
Minimum Value	`100`
Maximum Value (32-bit platforms)	`2**32-1`
Maximum Value (Unix, 64-bit platforms)	`2**64-1`
Maximum Value (Windows, 64-bit platforms)	`2**32-1`

If flushing activity falls behind, `InnoDB` can flush more aggressively, at a higher rate of I/O operations per second (IOPS) than defined by the `innodb_io_capacity` variable. The `innodb_io_capacity_max` variable defines a maximum number of IOPS performed by `InnoDB` background tasks in such situations.

For information about configuring the `innodb_io_capacity_max` variable, see Section 15.8.7, "Configuring InnoDB I/O Capacity".

- `innodb_limit_optimistic_insert_debug`

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

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

- `innodb_lock_wait_timeout`

Command-Line Format	`--innodb-lock-wait-timeout=#`

System Variable	innodb_lock_wait_timeout
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	50
Minimum Value	1
Maximum Value	1073741824
Unit	seconds

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

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

When a lock wait timeout occurs, the current statement is rolled back (not the entire transaction). To have the entire transaction roll back, start the server with the `--innodb-rollback-on-timeout` option. See also Section 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` applies to InnoDB row locks. A MySQL table lock does not happen inside InnoDB and this timeout does not apply to waits for table locks.

The lock wait timeout value does not apply to deadlocks when `innodb_deadlock_detect` is enabled (the default) because InnoDB detects deadlocks immediately and rolls back one of the deadlocked transactions. When `innodb_deadlock_detect` is disabled, InnoDB relies on `innodb_lock_wait_timeout` for transaction rollback when a deadlock occurs. See Section 15.7.5.2, “Deadlock Detection”.

`innodb_lock_wait_timeout` can be set at runtime with the `SET GLOBAL` or `SET SESSION` statement. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, “System Variable Privileges”) and affects the operation of all clients that subsequently connect. Any client can change the `SESSION` setting for `innodb_lock_wait_timeout`, which affects only that client.

- `innodb_log_buffer_size`

Command-Line Format	--innodb-log-buffer-size=#
System Variable	innodb_log_buffer_size
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	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

Command-Line Format	--innodb-log-checkpoint-fuzzy-now[={OFF|ON}]
Introduced	8.0.13
System Variable	innodb_log_checkpoint_fuzzy_now
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean
Default Value	OFF

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

- innodb_log_checkpoint_now

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

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

- innodb_log_checksums

Command-Line Format	--innodb-log-checksums[={OFF|ON}]
System Variable	innodb_log_checksums
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean

Default Value	ON

Enables or disables checksums for redo log pages.

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

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

- innodb_log_compressed_pages

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

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

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

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

For related information, see Section 15.9.1.6, “Compression for OLTP Workloads”.

- innodb_log_file_size

Command-Line Format	--innodb-log-file-size=#
Deprecated	8.0.30
System Variable	innodb_log_file_size
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	50331648
Minimum Value	4194304
Maximum Value	512GB / innodb_log_files_in_group

Unit	bytes

Note

`innodb_log_file_size` and `innodb_log_files_in_group` 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”.

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

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

The minimum `innodb_log_file_size` is 4MB.

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

If `innodb_dedicated_server` is enabled, the `innodb_log_file_size` value is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_log_files_in_group`

Command-Line Format	`--innodb-log-files-in-group=#`
Deprecated	8.0.30
System Variable	`innodb_log_files_in_group`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`2`
Minimum Value	`2`
Maximum Value	`100`

Note

`innodb_log_file_size` and `innodb_log_files_in_group` 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”.

The number of log files in the log group. `InnoDB` writes to the files in a circular fashion. The default (and recommended) value is 2. The location of the files is specified by

`innodb_log_group_home_dir`. The combined size of log files (`innodb_log_file_size` * `innodb_log_files_in_group`) can be up to 512GB.

For related information, see Redo Log Configuration.

If `innodb_dedicated_server` is enabled, `innodb_log_files_in_group` is automatically configured if it is not explicitly defined. For more information, see Section 15.8.12, “Enabling Automatic Configuration for a Dedicated MySQL Server”.

- `innodb_log_group_home_dir`

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

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

For related information, see Redo Log Configuration.

- `innodb_log_spin_cpu_abs_lwm`

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

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

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_spin_cpu_pct_hwm`

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

Maximum Value	`100`

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

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

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_wait_for_flush_spin_hwm`

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

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

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

to the operating system or file system due to a mismatch between write-ahead block size for the redo log and operating system or file system cache block size.

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

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

For related information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_log_writer_threads`

Command-Line Format	`--innodb-log-writer-threads[={OFF|ON}]`
Introduced	8.0.22
System Variable	`innodb_log_writer_threads`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Boolean
Default Value	`ON`

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

For more information, see Section 8.5.4, “Optimizing InnoDB Redo Logging”.

- `innodb_lru_scan_depth`

Command-Line Format	`--innodb-lru-scan-depth=#`
System Variable	`innodb_lru_scan_depth`
Scope	Global
Dynamic	Yes
`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`, start with a low value and configure the setting upward with the goal of rarely seeing zero free pages. Also, consider adjusting `innodb_lru_scan_depth` when changing the number of buffer pool instances, since `innodb_lru_scan_depth` * `innodb_buffer_pool_instances` defines the amount of work performed by the page cleaner thread each second.

For related information, see Section 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`

Command-Line Format	`--innodb-max-dirty-pages-pct=#`
System Variable	`innodb_max_dirty_pages_pct`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Numeric
Default Value	`90`
Minimum Value	`0`
Maximum Value	`99.99`

`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` 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`

Command-Line Format	`--innodb-max-dirty-pages-pct-lwm=#`
System Variable	`innodb_max_dirty_pages_pct_lwm`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Numeric
Default Value	`10`
Minimum Value	`0`
Maximum Value	`99.99`

Defines a low water mark representing the percentage of dirty pages at which preflushing is enabled to control the dirty page ratio. A value of 0 disables the pre-flushing behavior entirely. The

configured value should always be lower than the `innodb_max_dirty_pages_pct` value. For more information, see Section 15.8.3.5, “Configuring Buffer Pool Flushing”.

- `innodb_max_purge_lag`

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

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

For more information, see Section 15.8.9, “Purge Configuration”.

- `innodb_max_purge_lag_delay`

Command-Line Format	`--innodb-max-purge-lag-delay=#`
System Variable	`innodb_max_purge_lag_delay`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`10000000`
Unit	microseconds

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

For more information, see Section 15.8.9, “Purge Configuration”.

- `innodb_max_undo_log_size`

Command-Line Format	`--innodb-max-undo-log-size=#`
System Variable	`innodb_max_undo_log_size`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1073741824`
Minimum Value	`10485760`

Maximum Value	`2**64-1`
Unit	bytes

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

For more information, see Truncating Undo Tablespaces.

- `innodb_merge_threshold_set_all_debug`

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

Defines a page-full percentage value for index pages that overrides the current `MERGE_THRESHOLD` setting for all indexes that are currently in the dictionary cache. This option is only available if debugging support is compiled in using the `WITH_DEBUG CMake` option. For related information, see Section 15.8.11, “Configuring the Merge Threshold for Index Pages”.

- `innodb_monitor_disable`

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

This variable acts as a switch, disabling `InnoDB` metrics counters. Counter data may be queried using the Information Schema `INNODB_METRICS` table. For usage information, see Section 15.15.6, “InnoDB INFORMATION_SCHEMA Metrics Table”.

`innodb_monitor_disable='latch'` disables statistics collection for `SHOW ENGINE INNODB MUTEX`. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement”.

- `innodb_monitor_enable`

Command-Line Format	`--innodb-monitor-enable={counter|module|pattern|all}`
System Variable	`innodb_monitor_enable`
Scope	Global
Dynamic	Yes
`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, “InnoDB INFORMATION_SCHEMA Metrics Table”.

innodb_monitor_enable='latch' enables statistics collection for SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement”.

- innodb_monitor_reset

Command-Line Format	--innodb-monitor-reset={counter|module|pattern|all}
System Variable	innodb_monitor_reset
Scope	Global
Dynamic	Yes
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”.

innodb_monitor_reset='latch' resets statistics reported by SHOW ENGINE INNODB MUTEX. For more information, see Section 13.7.7.15, “SHOW ENGINE Statement”.

- innodb_monitor_reset_all

Command-Line Format	--innodb-monitor-reset-all={counter|module|pattern|all}
System Variable	innodb_monitor_reset_all
Scope	Global
Dynamic	Yes
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 Table”.

- `innodb_numa_interleave`

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

Enables the NUMA interleave memory policy for allocation of the `InnoDB` buffer pool. When `innodb_numa_interleave` is enabled, the NUMA memory policy is set to `MPOL_INTERLEAVE` for the `mysqld` process. After the `InnoDB` buffer pool is allocated, the NUMA memory policy is set back to `MPOL_DEFAULT`. For the `innodb_numa_interleave` option to be available, MySQL must be compiled on a NUMA-enabled Linux system.

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

- `innodb_old_blocks_pct`

Command-Line Format	`--innodb-old-blocks-pct=#`
System Variable	`innodb_old_blocks_pct`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`37`
Minimum Value	`5`
Maximum Value	`95`

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

For more information, see Section 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`

Command-Line Format	`--innodb-old-blocks-time=#`
System Variable	`innodb_old_blocks_time`
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	1000
Minimum Value	0
Maximum Value	2**32-1
Unit	milliseconds

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

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

The default value is 1000.

This variable is often used in combination with innodb_old_blocks_pct. For more information, see Section 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

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

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

- innodb_open_files

Command-Line Format	--innodb-open-files=#

System Variable	innodb_open_files
Scope	Global
Dynamic (≥ 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 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 limit can be set at runtime using a SELECT innodb_set_open_files_limit(*N*) statement, where *N* is the desired innodb_open_files limit; for example:

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

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

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

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

As of MySQL 8.0.28, to prevent non-LRU manged files from consuming the entire innodb_open_files limit, non-LRU managed files are limited to 90 percent of the innodb_open_files limit, which reserves 10 percent of the innodb_open_files limit for LRU managed files.

Temporary tablespace files were not counted toward the innodb_open_files limit from MySQL 8.0.24 to MySQL 8.0.27.

- innodb_optimize_fulltext_only

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

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

By default, OPTIMIZE TABLE reorganizes data in the clustered index of the table. When this option is enabled, OPTIMIZE TABLE skips the reorganization of table data, and instead processes newly

added, deleted, and updated token data for `InnoDB FULLTEXT` indexes. For more information, see Optimizing InnoDB Full-Text Indexes.

- `innodb_page_cleaners`

Command-Line Format	`--innodb-page-cleaners=#`
System Variable	`innodb_page_cleaners`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`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` default value is 4. If the number of page cleaner threads exceeds the number of buffer pool instances, `innodb_page_cleaners` is automatically set to the same value as `innodb_buffer_pool_instances`.

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

Multithreaded page cleaner support extends to shutdown and recovery phases.

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

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

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

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

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

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

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

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

```
systemctl restart mysql   # Debian platforms
```

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

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

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

- `innodb_page_size`

Command-Line Format	`--innodb-page-size=#`
System Variable	`innodb_page_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`16384`

Valid Values	`4096`
	`8192`
	`16384`
	`32768`
	`65536`

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

`innodb_page_size` can only be configured prior to initializing the MySQL instance and cannot be changed afterward. If no value is specified, the instance is initialized using the default page size. See Section 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` is set to 32KB or 64KB. For `innodb_page_size=32KB`, extent size is 2MB. For `innodb_page_size=64KB`, extent size is 4MB. `innodb_log_buffer_size` should be set to at least 16M (the default) when using 32KB or 64KB page sizes.

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

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

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

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

- `innodb_parallel_read_threads`

Command-Line Format	`--innodb-parallel-read-threads=#`
Introduced	8.0.14
System Variable	`innodb_parallel_read_threads`
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` session variable must be set to a value greater than 1 for parallel clustered index reads to occur. The actual number of threads used to perform a parallel clustered index read is determined by the `innodb_parallel_read_threads` setting or the number of index subtrees to scan, whichever is smaller. The pages read into the buffer pool during the scan are kept at the tail of the buffer pool LRU list so that they can be discarded quickly when free buffer pool pages are required.

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`

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

When this option is enabled, information about all deadlocks in `InnoDB` user transactions is recorded in the `mysqld` error log. Otherwise, you see information about only the last deadlock, using the `SHOW ENGINE INNODB STATUS` 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`

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

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

- `innodb_purge_batch_size`

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

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

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

For related information, see Section 15.8.9, “Purge Configuration”.

- `innodb_purge_threads`

Command-Line Format	`--innodb-purge-threads=#`
System Variable	`innodb_purge_threads`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`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

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

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

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

- innodb_random_read_ahead

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

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

For details about performance considerations for different types of read-ahead requests, see Section 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

Command-Line Format	--innodb-read-ahead-threshold=#
System Variable	innodb_read_ahead_threshold
Scope	Global

Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	56
Minimum Value	0
Maximum Value	64

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

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

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

For more information, see Section 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

Command-Line Format	--innodb-read-io-threads=#
System Variable	innodb_read_io_threads
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	4
Minimum Value	1
Maximum Value	64

The number of I/O threads for read operations in InnoDB. Its counterpart for write threads is innodb_write_io_threads. For more information, see Section 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, innodb_write_io_threads, and the Linux aio-max-nr setting can exceed system limits. Ideally, increase the aio-max-nr setting; as a

workaround, you might reduce the settings for one or both of the MySQL variables.

- `innodb_read_only`

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

Starts `InnoDB` in read-only mode. For distributing database applications or data sets on read-only media. Can also be used in data warehouses to share the same data directory between multiple instances. For more information, see Section 15.8.2, “Configuring InnoDB for Read-Only Operation”.

Previously, enabling the `innodb_read_only` system variable prevented creating and dropping tables only for the `InnoDB` storage engine. As of MySQL 8.0, enabling `innodb_read_only` 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` is enabled. The same principle applies to other table operations that require modifying data dictionary tables. Examples:

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

In addition, other tables in the `mysql` system database use the `InnoDB` storage engine 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`

Command-Line Format	`--innodb-redo-log-archive-dirs`
Introduced	8.0.17
System Variable	`innodb_redo_log_archive_dirs`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	String

Default Value	NULL

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

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

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

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

- innodb_redo_log_capacity

Command-Line Format	--innodb-redo-log-capacity=#
Introduced	8.0.30
System Variable	innodb_redo_log_capacity
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	104857600
Minimum Value	8388608
Maximum Value	137438953472
Unit	bytes

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

This variable supersedes the innodb_log_files_in_group and innodb_log_file_size 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 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

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

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

- `innodb_replication_delay`

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

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

- `innodb_rollback_on_timeout`

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

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

For more information, see Section 15.21.5, “InnoDB Error Handling”.

- `innodb_rollback_segments`

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

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

Command-Line Format	`--innodb-saved-page-number-debug=#`
System Variable	`innodb_saved_page_number_debug`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`0`
Minimum Value	`0`
Maximum Value	`2**23-1`

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

- `innodb_segment_reserve_factor`

Command-Line Format	`--innodb-segment-reserve-factor=#`
Introduced	8.0.26
System Variable	`innodb_segment_reserve_factor`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Numeric
Default Value	`12.5`
Minimum Value	`0.03`
Maximum Value	`40`

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

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

- `innodb_sort_buffer_size`

Command-Line Format	`--innodb-sort-buffer-size=#`
System Variable	`innodb_sort_buffer_size`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`1048576`
Minimum Value	`65536`

Maximum Value	67108864
Unit	bytes

This variable defines:

- The 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 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”.

- innodb_spin_wait_delay

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

Command-Line Format	--innodb-spin-wait-pause-multiplier=#
Introduced	8.0.16
System Variable	innodb_spin_wait_pause_multiplier
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

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

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

For more information, see Section 15.8.10.1, “Configuring Persistent Optimizer Statistics Parameters”.

- innodb_stats_include_delete_marked

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

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

`innodb_stats_include_delete_marked` can be enabled to ensure that `InnoDB` includes delete-marked records when calculating persistent optimizer statistics.

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

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

For related information, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters".

- `innodb_stats_method`

Command-Line Format	`--innodb-stats-method=value`
System Variable	`innodb_stats_method`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Enumeration
Default Value	`nulls_equal`

Valid Values	`nulls_equal`
	`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`

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

This option only applies when optimizer statistics are configured to be non-persistent. Optimizer statistics are not persisted to disk when `innodb_stats_persistent` is disabled or when individual tables are created or altered with `STATS_PERSISTENT=0`. For more information, see Section 15.8.10.2, "Configuring Non-Persistent Optimizer Statistics Parameters".

When `innodb_stats_on_metadata` is enabled, `InnoDB` updates non-persistent statistics when metadata statements such as `SHOW TABLE STATUS` or when accessing the Information Schema

TABLES or STATISTICS tables. (These updates are similar to what happens for ANALYZE TABLE.) When disabled, InnoDB does not update statistics during these operations. Leaving the setting disabled can improve access speed for schemas that have a large number of tables or indexes. It can also improve the stability of execution plans for queries that involve InnoDB tables.

To change the setting, issue the statement SET GLOBAL innodb_stats_on_metadata=*mode*, where *mode* is either ON or OFF (or 1 or 0). Changing the setting requires privileges sufficient to set global system variables (see Section 5.1.9.1, "System Variable Privileges") and immediately affects the operation of all connections.

- innodb_stats_persistent

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

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

For more information, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters".

- innodb_stats_persistent_sample_pages

Command-Line Format	--innodb-stats-persistent-sample-pages=#
System Variable	innodb_stats_persistent_sample_pages
Scope	Global
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Integer
Default Value	20
Minimum Value	1
Maximum Value	18446744073709551615

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by ANALYZE TABLE. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O during the execution of ANALYZE TABLE for an InnoDB table. For more information, see Section 15.8.10.1, "Configuring Persistent Optimizer Statistics Parameters".

Note

Setting a high value for innodb_stats_persistent_sample_pages could result in lengthy ANALYZE TABLE execution time. To estimate the number of database pages accessed by ANALYZE TABLE, see

Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

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

- `innodb_stats_transient_sample_pages`

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

The number of index pages to sample when estimating cardinality and other statistics for an indexed column, such as those calculated by `ANALYZE TABLE`. The default value is 8. Increasing the value improves the accuracy of index statistics, which can improve the query execution plan, at the expense of increased I/O when opening an `InnoDB` table or recalculating statistics. For more information, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

Note

Setting a high value for `innodb_stats_transient_sample_pages` could result in lengthy `ANALYZE TABLE` execution time. To estimate the number of database pages accessed by `ANALYZE TABLE`, see Section 15.8.10.3, “Estimating ANALYZE TABLE Complexity for InnoDB Tables”.

`innodb_stats_transient_sample_pages` only applies when `innodb_stats_persistent` is disabled for a table; when `innodb_stats_persistent` is enabled, `innodb_stats_persistent_sample_pages` applies instead. Takes the place of `innodb_stats_sample_pages`. For more information, see Section 15.8.10.2, “Configuring Non-Persistent Optimizer Statistics Parameters”.

- `innodb_status_output`

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

Enables or disables periodic output for the standard `InnoDB` Monitor. Also used in combination with `innodb_status_output_locks` to enable or disable periodic output for the `InnoDB` Lock Monitor. For more information, see Section 15.17.2, “Enabling InnoDB Monitors”.

- `innodb_status_output_locks`

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

Enables or disables the `InnoDB` Lock Monitor. When enabled, the `InnoDB` Lock Monitor prints additional information about locks in `SHOW ENGINE INNODB STATUS` output and in periodic output printed to the MySQL error log. Periodic output for the `InnoDB` Lock Monitor is printed as part of the standard `InnoDB` Monitor output. The standard `InnoDB` Monitor must therefore be enabled for the `InnoDB` Lock Monitor to print data to the MySQL error log periodically. For more information, see Section 15.17.2, “Enabling InnoDB Monitors”.

- `innodb_strict_mode`

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

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

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

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

You can enable or disable `innodb_strict_mode` on the command line when starting `mysqld`, or in a MySQL configuration file. You can also enable or disable `innodb_strict_mode` at runtime with the statement `SET [GLOBAL|SESSION] innodb_strict_mode=`*`mode`*, where *`mode`* is either `ON` or `OFF`. Changing the `GLOBAL` setting requires privileges sufficient to set global system variables (see Section 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`, 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`

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

Dynamic	No
SET_VAR Hint Applies	No
Type	Integer
Default Value	1
Minimum Value	1
Maximum Value	1024

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

- innodb_sync_spin_loops

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

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

- innodb_sync_debug

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

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

- innodb_table_locks

Command-Line Format	--innodb-table-locks[={OFF|ON}]
System Variable	innodb_table_locks
Scope	Global, Session
Dynamic	Yes
SET_VAR Hint Applies	No
Type	Boolean

Default Value	ON

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

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

For related information, see Section 15.7, “InnoDB Locking and Transaction Model”.

- innodb_temp_data_file_path

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

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

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

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

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

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

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

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

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

```
[mysqld]
```

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

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

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

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

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

For more information, see Global Temporary Tablespace.

- `innodb_temp_tablespaces_dir`

Command-Line Format	`--innodb-temp-tablespaces-dir=dir_name`
Introduced	8.0.13
System Variable	`innodb_temp_tablespaces_dir`
Scope	Global
Dynamic	No
`SET_VAR` Hint Applies	No
Type	Directory name
Default Value	`#innodb_temp`

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

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`

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

Maximum Value	`1000`

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

`InnoDB` tries to keep the number of threads inside `InnoDB` less than or equal to the `innodb_thread_concurrency` limit. Once the limit is reached, additional threads are placed into a “First In, First Out” (FIFO) queue for waiting threads. Threads waiting for locks are not counted in the number of concurrently executing threads.

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

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

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

Note

From an operating system perspective, using a resource management solution to manage how CPU time is shared among applications may be preferable to binding the `mysqld` process. For example, you could assign 90% of virtual CPU time to a given application while other critical processes *are not* running, and scale that value back to 40% when other critical processes *are* running.

- In some cases, the optimal `innodb_thread_concurrency` setting can be smaller than the number of virtual CPUs.
- An `innodb_thread_concurrency` value that is too high can cause performance regression due to increased contention on system internals and resources.
- Monitor and analyze your system regularly. Changes to workload, number of users, or computing environment may require that you adjust the `innodb_thread_concurrency` setting.

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

For related information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- `innodb_thread_sleep_delay`

Command-Line Format	`--innodb-thread-sleep-delay=#`
System Variable	`innodb_thread_sleep_delay`
Scope	Global
Dynamic	Yes
`SET_VAR` Hint Applies	No
Type	Integer
Default Value	`10000`
Minimum Value	`0`
Maximum Value	`1000000`
Unit	microseconds

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

For more information, see Section 15.8.4, “Configuring Thread Concurrency for InnoDB”.

- `innodb_tmpdir`

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

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

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

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

TABLE operation reports an error if innodb_tmpdir is set to an invalid directory. innodb_tmpdir overrides the MySQL tmpdir setting but only for online ALTER TABLE operations.

The FILE privilege is required to configure innodb_tmpdir.

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

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

For more information, see Section 15.12.3, "Online DDL Space Requirements". For information about online ALTER TABLE operations, see Section 15.12, "InnoDB and Online DDL".

- innodb_trx_purge_view_update_only_debug

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

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

- innodb_trx_rseg_n_slots_debug

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

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

- innodb_undo_directory

Command-Line Format	--innodb-undo-directory=dir_name

System Variable	innodb_undo_directory
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Directory name

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

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

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

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

For more information, see Section 15.6.3.4, “Undo Tablespaces”.

- innodb_undo_log_encrypt

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

Controls encryption of undo log data for tables encrypted using the InnoDB data-at-rest encryption feature. Only applies to undo logs that reside in separate undo tablespaces. See Section 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.

- innodb_undo_log_truncate

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

When enabled, undo tablespaces that exceed the threshold value defined by innodb_max_undo_log_size are marked for truncation. Only undo tablespaces can be truncated.

Truncating undo logs that reside in the system tablespace is not supported. For truncation to occur, there must be at least two undo tablespaces.

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

For more information, see Truncating Undo Tablespaces.

- `innodb_undo_tablespaces`

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

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

Note

The `innodb_undo_tablespaces` variable is deprecated 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`

Command-Line Format	`--innodb-use-fdatasync[={OFF|ON}]`
Introduced	8.0.26
System Variable	`innodb_use_fdatasync`
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` 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` settings such as `fsync`, `O_DSYNC`, and `O_DIRECT` use `fsync()` system calls. The `innodb_use_fdatasync` variable is applicable when using those settings.

- `innodb_use_native_aio`

Command-Line Format	`--innodb-use-native-aio[={OFF|ON}]`

System Variable	innodb_use_native_aio
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

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

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

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

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

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

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

For more information, see Section 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
Scope	Global
Dynamic	No
SET_VAR Hint Applies	No
Type	Boolean
Default Value	ON

Controls tablespace file path validation. At startup, InnoDB validates the paths of known tablespace files against tablespace file paths stored in the data dictionary in case tablespace files have been moved to a different location. The innodb_validate_tablespace_paths variable permits disabling tablespace path validation. This feature is intended for environments where tablespaces

files are not moved. Disabling path validation improves startup time on systems with a large number of tablespace files.

Warning

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

For more information, see Section 15.6.3.7, “Disabling Tablespace Path Validation”.

- `innodb_version`

  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`

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

The number of I/O threads for write operations in `InnoDB`. The default value is 4. Its counterpart for read threads is `innodb_read_io_threads`. For more information, see Section 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`, `innodb_write_io_threads`, and the Linux `aio-max-nr` setting can exceed system limits. Ideally, increase the `aio-max-nr` setting; as a workaround, you might reduce the settings for one or both of the MySQL variables.

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

For general I/O tuning advice, see Section 8.5.8, “Optimizing InnoDB Disk I/O”.

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”. 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 Page Compression” 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, “InnoDB Table and Page Compression”, 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 Transaction and Locking Information, in MySQL 5.7 Reference Manual.

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 Information, in MySQL 5.7 Reference Manual.

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, “InnoDB INFORMATION_SCHEMA Transaction and Locking Information”.)

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 Session Becomes Idle.

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	2008-01-15 16:44:54	NULL	NULL	2	5	SELECT SLEEP(100)
A4	LOCK WAIT	2008-01-15 16:45:09	A4:1:3:2	2008-01-15 16:45:09	2	6	SELECT b FROM t FOR UPDATE
A5	LOCK WAIT	2008-01-15 16:45:14	A5:1:3:2	2008-01-15 16:45:14	2	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;
   ```

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 Transaction and Locking Information”.

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 PROCESSLIST
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	2008-01-15 13:10:16	77F	2008-01-15 13:10:16	1	876	INSERT INTO t09 (D, B, C) VALUES …
77E	LOCK WAIT	2008-01-15 13:10:16	77E	2008-01-15 13:10:16	1	875	INSERT INTO t09 (D, B, C) VALUES …
77D	LOCK WAIT	2008-01-15 13:10:16	77D	2008-01-15 13:10:16	1	874	INSERT INTO t09 (D, B, C) VALUES …
77B	LOCK WAIT	2008-01-15 13:10:16	77B:733:12:1	2008-01-15 13:10:16	4	873	INSERT INTO t09 (D, B, C) VALUES …
77A	RUNNING	2008-01-15 13:10:16	NULL	NULL	4	872	SELECT b, c FROM t09 WHERE …
E56	LOCK WAIT	2008-01-15 13:10:06	E56:743:6:2	2008-01-15 13:10:06	5	384	INSERT INTO t2 VALUES …
E55	LOCK WAIT	2008-01-15 13:10:06	E55:743:38:2	2008-01-15 13:10:13	965	257	INSERT INTO t2 VALUES …
19C	RUNNING	2008-01-15 13:09:10	NULL	NULL	2900	130	INSERT INTO t2 VALUES …
E15	RUNNING	2008-01-15 13:08:59	NULL	NULL	5395	61	INSERT INTO t2 VALUES …
51D	RUNNING	2008-01-15 13:08:47	NULL	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:733:12:1	77B	X	RECORD	test	t09	PRIMARY	supremum pseudo-record
77A:733:12:1	77A	X	RECORD	test	t09	PRIMARY	supremum pseudo-record
E56:743:6:2	E56	S	RECORD	test	t2	PRIMARY	0, 0
E55:743:6:2	E55	X	RECORD	test	t2	PRIMARY	0, 0
E55:743:38:2	E55	S	RECORD	test	t2	PRIMARY	1922, 1922
19C:743:38:2	19C	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, in MySQL 5.7 Reference Manual.

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 Transaction and Locking Information, in MySQL 5.7 Reference Manual.

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”, 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
```

```
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))
       ENGINE = InnoDB;

mysql> CREATE INDEX i1 ON t1(col1);
```

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
 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

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)
       ON DELETE CASCADE) ENGINE=INNODB;
```

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

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/%'
       ORDER BY a.NAME DESC;
+----------------------+------------+-----------+-------+-----------+
| NAME                 | ROW_FORMAT | page_size | pk_mb | secidx_mb |
+----------------------+------------+-----------+-------+-----------+
| employees/titles     | Dynamic    |     16384 |    20 |        11 |
| employees/salaries   | Dynamic    |     16384 |    93 |        34 |
| employees/employees  | Dynamic    |     16384 |    15 |         0 |
| employees/dept_manager | Dynamic  |     16384 |     0 |         0 |
| employees/dept_emp   | Dynamic    |     16384 |    12 |        10 |
| employees/departments | Dynamic   |     16384 |     0 |         0 |
+----------------------+------------+-----------+-------+-----------+
```

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` or `innodb_ft_total_cache_size` 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` 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)
```

```
       ) 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 ...'),
       ('MySQL Security','When configured properly, MySQL ...');
```

2. Set the `innodb_ft_aux_table` 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 |
+------------+--------------+-------------+-----------+--------+----------+
| 1001       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

4. Enable the `innodb_optimize_fulltext_only` 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` 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       |            5 |           5 |         1 |      5 |        0 |
| after      |            3 |           3 |         1 |      3 |       22 |
| comparison |            6 |           6 |         1 |      6 |       44 |
| configured |            7 |           7 |         1 |      7 |       20 |
| database   |            2 |           6 |         2 |      2 |       31 |
+------------+--------------+-------------+-----------+--------+----------+
```

   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 | 180   |
| synced_doc_id             | 8     |
| stopword_table_name       |       |
| use_stopword              | 1     |
+---------------------------+-------+
```

10. Disable `innodb_optimize_fulltext_only`, 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:

```
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);
+--------------------+
| PAGE_TYPE          |
+--------------------+
| SYSTEM             |
| IBUF_BITMAP        |
| UNKNOWN            |
| FILE_SPACE_HEADER  |
| INODE              |
| UNDO_LOG           |
| ALLOCATED          |
+--------------------+
```

Example 15.7 Querying User Data in the INNODB_BUFFER_PAGE Table

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
       WHERE TABLE_NAME IS NOT NULL AND TABLE_NAME NOT LIKE '%INNODB_TABLES%';
+----------+
| COUNT(*) |
+----------+
|     7897 |
+----------+
```

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
       (SELECT COUNT(*) FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE
       WHERE TABLE_NAME IS NOT NULL AND (INSTR(TABLE_NAME, '/') > 0 OR INSTR(TABLE_NAME, '.') > 0)
       ) AS user_pages,
       (
       SELECT COUNT(*)
       FROM information_schema.INNODB_BUFFER_PAGE
       ) AS total_pages,
       (
       SELECT ROUND((user_pages/total_pages) * 100)
       ) AS user_page_percentage;
+------------+-------------+----------------------+
| user_pages | total_pages | user_page_percentage |
+------------+-------------+----------------------+
|       7897 |        8192 |                   96 |
+------------+-------------+----------------------+
```

This query identifies user-defined tables with pages in the buffer pool:

```
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)
       AND TABLE_NAME NOT LIKE '`mysql`.`innodb_%';
+-------------------------+
| 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     |  1608 |              25 |
| PRIMARY    |  6086 |              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 Standard Monitor and Lock Monitor Output".

```
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%';
+---------------------------------------+-------------+
| Variable_name                         | Value       |
+---------------------------------------+-------------+
| Innodb_buffer_pool_dump_status        | not started |
| Innodb_buffer_pool_load_status        | not started |
| Innodb_buffer_pool_resize_status      | not started |
| Innodb_buffer_pool_pages_data         | 8173        |
| Innodb_buffer_pool_bytes_data         | 133906432   |
| Innodb_buffer_pool_pages_dirty        | 0           |
| Innodb_buffer_pool_bytes_dirty        | 0           |
| Innodb_buffer_pool_pages_flushed      | 15109       |
| Innodb_buffer_pool_pages_free         | 1           |
| Innodb_buffer_pool_pages_misc         | 18          |
| Innodb_buffer_pool_pages_total        | 8192        |
| Innodb_buffer_pool_read_ahead_rnd     | 0           |
| Innodb_buffer_pool_read_ahead         | 2713        |
| Innodb_buffer_pool_read_ahead_evicted | 0           |
| Innodb_buffer_pool_read_requests      | 33069332    |
| Innodb_buffer_pool_reads              | 558         |
| Innodb_buffer_pool_wait_free          | 0           |
| Innodb_buffer_pool_write_requests     | 11985961    |
+---------------------------------------+-------------+
```

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_index | disabled |
| adaptive_hash_pages_removed            | adaptive_hash_index | disabled |
| adaptive_hash_rows_added               | adaptive_hash_index | disabled |
| adaptive_hash_rows_deleted_no_hash_entry | adaptive_hash_index | disabled |
| adaptive_hash_rows_removed             | adaptive_hash_index | disabled |
| adaptive_hash_rows_updated             | adaptive_hash_index | disabled |
| adaptive_hash_searches                 | adaptive_hash_index | enabled  |
| adaptive_hash_searches_btree           | adaptive_hash_index | enabled  |
| buffer_data_reads                      | buffer             | enabled  |
| buffer_data_written                    | buffer             | enabled  |
| buffer_flush_adaptive                  | buffer             | disabled |
| buffer_flush_adaptive_avg_pass         | buffer             | disabled |
| buffer_flush_adaptive_avg_time_est     | buffer             | disabled |
| buffer_flush_adaptive_avg_time_slot    | buffer             | disabled |
| buffer_flush_adaptive_avg_time_thread  | buffer             | disabled |
| buffer_flush_adaptive_pages            | buffer             | disabled |
| buffer_flush_adaptive_total_pages      | buffer             | disabled |
| buffer_flush_avg_page_rate             | buffer             | disabled |
| buffer_flush_avg_pass                  | buffer             | disabled |
| buffer_flush_avg_time                  | buffer             | disabled |
| buffer_flush_background                | buffer             | disabled |
| buffer_flush_background_pages          | buffer             | disabled |
| buffer_flush_background_total_pages    | buffer             | disabled |
| buffer_flush_batches                   | buffer             | disabled |
| buffer_flush_batch_num_scan            | buffer             | disabled |
| buffer_flush_batch_pages               | buffer             | disabled |
| buffer_flush_batch_scanned             | buffer             | disabled |
| buffer_flush_batch_scanned_per_call    | buffer             | disabled |
| buffer_flush_batch_total_pages         | buffer             | disabled |
| buffer_flush_lsn_avg_rate              | buffer             | disabled |
| buffer_flush_neighbor                  | buffer             | disabled |
| buffer_flush_neighbor_pages            | buffer             | disabled |
| buffer_flush_neighbor_total_pages      | buffer             | disabled |
| buffer_flush_n_to_flush_by_age         | buffer             | disabled |
| buffer_flush_n_to_flush_by_dirty_page  | buffer             | disabled |
| buffer_flush_n_to_flush_requested      | buffer             | disabled |
| buffer_flush_pct_for_dirty             | buffer             | disabled |
| buffer_flush_pct_for_lsn               | buffer             | disabled |
| buffer_flush_sync                      | buffer             | disabled |
| buffer_flush_sync_pages                | buffer             | disabled |
| buffer_flush_sync_total_pages          | buffer             | disabled |
| buffer_flush_sync_waits                | buffer             | disabled |
| buffer_LRU_batches_evict               | buffer             | disabled |
| buffer_LRU_batches_flush               | buffer             | disabled |
| buffer_LRU_batch_evict_pages           | buffer             | disabled |
| buffer_LRU_batch_evict_total_pages     | buffer             | disabled |
| buffer_LRU_batch_flush_avg_pass        | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_est    | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_slot   | buffer             | disabled |
| buffer_LRU_batch_flush_avg_time_thread | buffer             | disabled |
| buffer_LRU_batch_flush_pages           | buffer             | disabled |
| buffer_LRU_batch_flush_total_pages     | buffer             | disabled |
| buffer_LRU_batch_num_scan              | buffer             | disabled |
| buffer_LRU_batch_scanned               | buffer             | disabled |
| buffer_LRU_batch_scanned_per_call      | buffer             | disabled |
| buffer_LRU_get_free_loops              | buffer             | disabled |
| buffer_LRU_get_free_search             | Buffer             | disabled |
```

```
| buffer_LRU_get_free_waits                  | buffer             | disabled |
| buffer_LRU_search_num_scan                 | buffer             | disabled |
| buffer_LRU_search_scanned                  | buffer             | disabled |
| buffer_LRU_search_scanned_per_call         | buffer             | disabled |
| buffer_LRU_single_flush_failure_count      | Buffer             | disabled |
| buffer_LRU_single_flush_num_scan           | buffer             | disabled |
| buffer_LRU_single_flush_scanned            | buffer             | disabled |
| buffer_LRU_single_flush_scanned_per_call   | buffer             | disabled |
| buffer_LRU_unzip_search_num_scan           | buffer             | disabled |
| buffer_LRU_unzip_search_scanned            | buffer             | disabled |
| buffer_LRU_unzip_search_scanned_per_call   | buffer             | disabled |
| buffer_pages_created                       | buffer             | enabled  |
| buffer_pages_read                          | buffer             | enabled  |
| buffer_pages_written                       | buffer             | enabled  |
| buffer_page_read_blob                      | buffer_page_io     | disabled |
| buffer_page_read_fsp_hdr                   | buffer_page_io     | disabled |
| buffer_page_read_ibuf_bitmap               | buffer_page_io     | disabled |
| buffer_page_read_ibuf_free_list            | buffer_page_io     | disabled |
| buffer_page_read_index_ibuf_leaf           | buffer_page_io     | disabled |
| buffer_page_read_index_ibuf_non_leaf       | buffer_page_io     | disabled |
| buffer_page_read_index_inode               | buffer_page_io     | disabled |
| buffer_page_read_index_leaf                | buffer_page_io     | disabled |
| buffer_page_read_index_non_leaf            | buffer_page_io     | disabled |
| buffer_page_read_other                     | buffer_page_io     | disabled |
| buffer_page_read_rseg_array                | buffer_page_io     | disabled |
| buffer_page_read_system_page               | buffer_page_io     | disabled |
| buffer_page_read_trx_system                | buffer_page_io     | disabled |
| buffer_page_read_undo_log                  | buffer_page_io     | disabled |
| buffer_page_read_xdes                      | buffer_page_io     | disabled |
| buffer_page_read_zblob                     | buffer_page_io     | disabled |
| buffer_page_read_zblob2                    | buffer_page_io     | disabled |
| buffer_page_written_blob                   | buffer_page_io     | disabled |
| buffer_page_written_fsp_hdr                | buffer_page_io     | disabled |
| buffer_page_written_ibuf_bitmap            | buffer_page_io     | disabled |
| buffer_page_written_ibuf_free_list         | buffer_page_io     | disabled |
| buffer_page_written_index_ibuf_leaf        | buffer_page_io     | disabled |
| buffer_page_written_index_ibuf_non_leaf    | buffer_page_io     | disabled |
| buffer_page_written_index_inode            | buffer_page_io     | disabled |
| buffer_page_written_index_leaf             | buffer_page_io     | disabled |
| buffer_page_written_index_non_leaf         | buffer_page_io     | disabled |
| buffer_page_written_on_log_no_waits        | buffer_page_io     | disabled |
| buffer_page_written_on_log_waits           | buffer_page_io     | disabled |
| buffer_page_written_on_log_wait_loops      | buffer_page_io     | disabled |
| buffer_page_written_other                  | buffer_page_io     | disabled |
| buffer_page_written_rseg_array             | buffer_page_io     | disabled |
| buffer_page_written_system_page            | buffer_page_io     | disabled |
| buffer_page_written_trx_system             | buffer_page_io     | disabled |
| buffer_page_written_undo_log               | buffer_page_io     | disabled |
| buffer_page_written_xdes                   | buffer_page_io     | disabled |
| buffer_page_written_zblob                  | buffer_page_io     | disabled |
| buffer_page_written_zblob2                 | buffer_page_io     | disabled |
| buffer_pool_bytes_data                     | buffer             | enabled  |
| buffer_pool_bytes_dirty                    | buffer             | enabled  |
| buffer_pool_pages_data                     | buffer             | enabled  |
| buffer_pool_pages_dirty                    | buffer             | enabled  |
| buffer_pool_pages_free                     | buffer             | enabled  |
| buffer_pool_pages_misc                     | buffer             | enabled  |
| buffer_pool_pages_total                    | buffer             | enabled  |
| buffer_pool_reads                          | buffer             | enabled  |
| buffer_pool_read_ahead                     | buffer             | enabled  |
| buffer_pool_read_ahead_evicted             | buffer             | enabled  |
| buffer_pool_read_requests                  | buffer             | enabled  |
| buffer_pool_size                           | server             | enabled  |
| buffer_pool_wait_free                      | buffer             | enabled  |
| buffer_pool_write_requests                 | buffer             | enabled  |
| compression_pad_decrements                 | compression        | disabled |
| compression_pad_increments                 | compression        | disabled |
| compress_pages_compressed                  | compression        | disabled |
| compress_pages_decompressed                | compression        | disabled |
| cpu_n                                      | cpu                | disabled |
| cpu_stime_abs                              | cpu                | disabled |
| cpu_stime_pct                              | cpu                | disabled |
```

```
| cpu_utime_abs                          | cpu                | disabled |
| cpu_utime_pct                          | cpu                | disabled |
| dblwr_async_requests                   | dblwr              | disabled |
| dblwr_flush_requests                   | dblwr              | disabled |
| dblwr_flush_wait_events                | dblwr              | disabled |
| dblwr_sync_requests                    | dblwr              | disabled |
| ddl_background_drop_tables             | ddl                | disabled |
| ddl_log_file_alter_table               | ddl                | disabled |
| ddl_online_create_index                | ddl                | disabled |
| ddl_pending_alter_table                | ddl                | disabled |
| ddl_sort_file_alter_table              | ddl                | disabled |
| dml_deletes                            | dml                | enabled  |
| dml_inserts                            | dml                | enabled  |
| dml_reads                              | dml                | disabled |
| dml_system_deletes                     | dml                | enabled  |
| dml_system_inserts                     | dml                | enabled  |
| dml_system_reads                       | dml                | enabled  |
| dml_system_updates                     | dml                | enabled  |
| dml_updates                            | dml                | enabled  |
| file_num_open_files                    | file_system        | enabled  |
| ibuf_merges                            | change_buffer      | enabled  |
| ibuf_merges_delete                     | change_buffer      | enabled  |
| ibuf_merges_delete_mark                | change_buffer      | enabled  |
| ibuf_merges_discard_delete             | change_buffer      | enabled  |
| ibuf_merges_discard_delete_mark        | change_buffer      | enabled  |
| ibuf_merges_discard_insert             | change_buffer      | enabled  |
| ibuf_merges_insert                     | change_buffer      | enabled  |
| ibuf_size                              | change_buffer      | enabled  |
| icp_attempts                           | icp                | disabled |
| icp_match                              | icp                | disabled |
| icp_no_match                           | icp                | disabled |
| icp_out_of_range                       | icp                | disabled |
| index_page_discards                    | index              | disabled |
| index_page_merge_attempts              | index              | disabled |
| index_page_merge_successful            | index              | disabled |
| index_page_reorg_attempts              | index              | disabled |
| index_page_reorg_successful            | index              | disabled |
| index_page_splits                      | index              | disabled |
| innodb_activity_count                  | server             | enabled  |
| innodb_background_drop_table_usec      | server             | disabled |
| innodb_dblwr_pages_written             | server             | enabled  |
| innodb_dblwr_writes                    | server             | enabled  |
| innodb_dict_lru_count                  | server             | disabled |
| innodb_dict_lru_usec                   | server             | disabled |
| innodb_ibuf_merge_usec                 | server             | disabled |
| innodb_master_active_loops             | server             | disabled |
| innodb_master_idle_loops               | server             | disabled |
| innodb_master_purge_usec               | server             | disabled |
| innodb_master_thread_sleeps            | server             | disabled |
| innodb_mem_validate_usec               | server             | disabled |
| innodb_page_size                       | server             | enabled  |
| innodb_rwlock_sx_os_waits              | server             | enabled  |
| innodb_rwlock_sx_spin_rounds           | server             | enabled  |
| innodb_rwlock_sx_spin_waits            | server             | enabled  |
| innodb_rwlock_s_os_waits               | server             | enabled  |
| innodb_rwlock_s_spin_rounds            | server             | enabled  |
| innodb_rwlock_s_spin_waits             | server             | enabled  |
| innodb_rwlock_x_os_waits               | server             | enabled  |
| innodb_rwlock_x_spin_rounds            | server             | enabled  |
| innodb_rwlock_x_spin_waits             | server             | enabled  |
| lock_deadlocks                         | lock               | enabled  |
| lock_deadlock_false_positives          | lock               | enabled  |
| lock_deadlock_rounds                   | lock               | enabled  |
| lock_rec_grant_attempts                | lock               | enabled  |
| lock_rec_locks                         | lock               | disabled |
| lock_rec_lock_created                  | lock               | disabled |
| lock_rec_lock_removed                  | lock               | disabled |
| lock_rec_lock_requests                 | lock               | disabled |
| lock_rec_lock_waits                    | lock               | disabled |
| lock_rec_release_attempts              | lock               | enabled  |
| lock_row_lock_current_waits            | lock               | enabled  |
| lock_row_lock_time                     | lock               | enabled  |
```

```
| lock_row_lock_time_avg                | lock               | enabled  |
| lock_row_lock_time_max                | lock               | enabled  |
| lock_row_lock_waits                   | lock               | enabled  |
| lock_schedule_refreshes               | lock               | enabled  |
| lock_table_locks                      | lock               | disabled |
| lock_table_lock_created               | lock               | disabled |
| lock_table_lock_removed               | lock               | disabled |
| lock_table_lock_waits                 | lock               | disabled |
| lock_threads_waiting                  | lock               | enabled  |
| lock_timeouts                         | lock               | enabled  |
| log_checkpoints                       | log                | disabled |
| log_concurrency_margin                | log                | disabled |
| log_flusher_no_waits                  | log                | disabled |
| log_flusher_waits                     | log                | disabled |
| log_flusher_wait_loops                | log                | disabled |
| log_flush_avg_time                    | log                | disabled |
| log_flush_lsn_avg_rate                | log                | disabled |
| log_flush_max_time                    | log                | disabled |
| log_flush_notifier_no_waits           | log                | disabled |
| log_flush_notifier_waits              | log                | disabled |
| log_flush_notifier_wait_loops         | log                | disabled |
| log_flush_total_time                  | log                | disabled |
| log_free_space                        | log                | disabled |
| log_full_block_writes                 | log                | disabled |
| log_lsn_archived                      | log                | disabled |
| log_lsn_buf_dirty_pages_added         | log                | disabled |
| log_lsn_buf_pool_oldest_approx        | log                | disabled |
| log_lsn_buf_pool_oldest_lwm           | log                | disabled |
| log_lsn_checkpoint_age                | log                | disabled |
| log_lsn_current                       | log                | disabled |
| log_lsn_last_checkpoint               | log                | disabled |
| log_lsn_last_flush                    | log                | disabled |
| log_max_modified_age_async            | log                | disabled |
| log_max_modified_age_sync             | log                | disabled |
| log_next_file                         | log                | disabled |
| log_on_buffer_space_no_waits          | log                | disabled |
| log_on_buffer_space_waits             | log                | disabled |
| log_on_buffer_space_wait_loops        | log                | disabled |
| log_on_file_space_no_waits            | log                | disabled |
| log_on_file_space_waits               | log                | disabled |
| log_on_file_space_wait_loops          | log                | disabled |
| log_on_flush_no_waits                 | log                | disabled |
| log_on_flush_waits                    | log                | disabled |
| log_on_flush_wait_loops               | log                | disabled |
| log_on_recent_closed_wait_loops       | log                | disabled |
| log_on_recent_written_wait_loops      | log                | disabled |
| log_on_write_no_waits                 | log                | disabled |
| log_on_write_waits                    | log                | disabled |
| log_on_write_wait_loops               | log                | disabled |
| log_padded                            | log                | disabled |
| log_partial_block_writes              | log                | disabled |
| log_waits                             | log                | enabled  |
| log_writer_no_waits                   | log                | disabled |
| log_writer_on_archiver_waits          | log                | disabled |
| log_writer_on_file_space_waits        | log                | disabled |
| log_writer_waits                      | log                | disabled |
| log_writer_wait_loops                 | log                | disabled |
| log_writes                            | log                | enabled  |
| log_write_notifier_no_waits           | log                | disabled |
| log_write_notifier_waits              | log                | disabled |
| log_write_notifier_wait_loops         | log                | disabled |
| log_write_requests                    | log                | enabled  |
| log_write_to_file_requests_interval   | log                | disabled |
| metadata_table_handles_closed         | metadata           | disabled |
| metadata_table_handles_opened         | metadata           | disabled |
| metadata_table_reference_count        | metadata           | disabled |
| module_cpu                            | cpu                | disabled |
| module_dblwr                          | dblwr              | disabled |
| module_page_track                     | page_track         | disabled |
| os_data_fsyncs                        | os                 | enabled  |
| os_data_reads                         | os                 | enabled  |
| os_data_writes                        | os                 | enabled  |
```

```
| os_log_bytes_written                          | os                | enabled  |
| os_log_fsyncs                                 | os                | enabled  |
| os_log_pending_fsyncs                         | os                | enabled  |
| os_log_pending_writes                         | os                | enabled  |
| os_pending_reads                              | os                | disabled |
| os_pending_writes                             | os                | disabled |
| page_track_checkpoint_partial_flush_request | page_track          | disabled |
| page_track_full_block_writes                  | page_track        | disabled |
| page_track_partial_block_writes               | page_track        | disabled |
| page_track_resets                             | page_track        | disabled |
| purge_del_mark_records                        | purge             | disabled |
| purge_dml_delay_usec                          | purge             | disabled |
| purge_invoked                                 | purge             | disabled |
| purge_resume_count                            | purge             | disabled |
| purge_stop_count                              | purge             | disabled |
| purge_truncate_history_count                  | purge             | disabled |
| purge_truncate_history_usec                   | purge             | disabled |
| purge_undo_log_pages                          | purge             | disabled |
| purge_upd_exist_or_extern_records             | purge             | disabled |
| sampled_pages_read                            | sampling          | disabled |
| sampled_pages_skipped                         | sampling          | disabled |
| trx_active_transactions                       | transaction       | disabled |
| trx_allocations                               | transaction       | disabled |
| trx_commits_insert_update                     | transaction       | disabled |
| trx_nl_ro_commits                             | transaction       | disabled |
| trx_on_log_no_waits                           | transaction       | disabled |
| trx_on_log_waits                              | transaction       | disabled |
| trx_on_log_wait_loops                         | transaction       | disabled |
| trx_rollbacks                                 | transaction       | disabled |
| trx_rollbacks_savepoint                       | transaction       | disabled |
| trx_rollback_active                           | transaction       | disabled |
| trx_ro_commits                                | transaction       | disabled |
| trx_rseg_current_size                         | transaction       | disabled |
| trx_rseg_history_len                          | transaction       | enabled  |
| trx_rw_commits                                | transaction       | disabled |
| trx_undo_slots_cached                         | transaction       | disabled |
| trx_undo_slots_used                           | transaction       | disabled |
| undo_truncate_count                           | undo              | disabled |
| undo_truncate_done_logging_count              | undo              | disabled |
| undo_truncate_start_logging_count             | undo              | disabled |
| undo_truncate_usec                            | undo              | disabled |
+-----------------------------------------------+-------------------+----------+
314 rows in set (0.00 sec)
```

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` 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 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 85
     TIME_RESET: 2014-12-04 14:19:44
         STATUS: enabled
           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 14:18:28
  TIME_DISABLED: 2014-12-04 14:20:06
   TIME_ELAPSED: 98
     TIME_RESET: NULL
         STATUS: disabled
           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` 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 | ./mysql/plugin.ibd                    |
    |       3 | ./mysql/servers.ibd                   |
    |       4 | ./mysql/help_topic.ibd                |
    |       5 | ./mysql/help_category.ibd             |
    |       6 | ./mysql/help_relation.ibd             |
    |       7 | ./mysql/help_keyword.ibd              |
    |       8 | ./mysql/time_zone_name.ibd            |
    |       9 | ./mysql/time_zone.ibd                 |
    |      10 | ./mysql/time_zone_transition.ibd      |
    |      11 | ./mysql/time_zone_transition_type.ibd |
    |      12 | ./mysql/time_zone_leap_second.ibd     |
    |      13 | ./mysql/innodb_table_stats.ibd        |
    |      14 | ./mysql/innodb_index_stats.ibd        |
    |      15 | ./mysql/slave_relay_log_info.ibd      |
    |      16 | ./mysql/slave_master_info.ibd         |
    |      17 | ./mysql/slave_worker_info.ibd         |
    |      18 | ./mysql/gtid_executed.ibd             |
    |      19 | ./mysql/server_cost.ibd               |
    |      20 | ./mysql/engine_cost.ibd               |
    |      21 | ./sys/sys_config.ibd                  |
    |      23 | ./test/t1.ibd                         |
    |      26 | /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               | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex           | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex               | NO      | NO    |
...
| wait/io/file/innodb/innodb_data_file                    | YES     | YES   |
| wait/io/file/innodb/innodb_log_file                     | YES     | YES   |
| wait/io/file/innodb/innodb_temp_file                    | YES     | YES   |
| stage/innodb/alter table (end)                          | YES     | YES   |
| stage/innodb/alter table (flush)                        | YES     | YES   |
| stage/innodb/alter table (insert)                       | YES     | YES   |
| stage/innodb/alter table (log apply index)              | YES     | YES   |
| stage/innodb/alter table (log apply table)              | YES     | YES   |
| stage/innodb/alter table (merge sort)                   | YES     | YES   |
| stage/innodb/alter table (read PK and internal sort)    | YES     | YES   |
| stage/innodb/buffer pool load                           | YES     | YES   |
| memory/innodb/buf_buf_pool                              | NO      | NO    |
| memory/innodb/dict_stats_bg_recalc_pool_t               | NO      | NO    |
| memory/innodb/dict_stats_index_map_t                    | NO      | NO    |
| memory/innodb/dict_stats_n_diff_on_level                | NO      | NO    |
| memory/innodb/other                                     | NO      | NO    |
| memory/innodb/row_log_buf                               | NO      | NO    |
| memory/innodb/row_merge_sort                            | NO      | NO    |
```

```
| memory/innodb/std                                          | NO      | NO    |
| memory/innodb/sync_debug_latches                           | NO      | NO    |
| memory/innodb/trx_sys_t::rw_trx_ids                        | 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 InnoDB Tables Using Performance Schema”, 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

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'
       WHERE NAME LIKE 'stage/innodb/alter%';
Query OK, 7 rows affected (0.00 sec)
Rows matched: 7  Changed: 7  Warnings: 0
```

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%';
```

```
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
```

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
       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)
```

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 |
| stage/innodb/alter table (flush)                   |           1213 |           1213 |
| stage/innodb/alter table (log apply table)         |           1597 |           1597 |
| stage/innodb/alter table (end)                     |           1597 |           1597 |
| stage/innodb/alter table (log apply table)         |           1981 |           1981 |
+----------------------------------------------------+----------------+----------------+
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%';
```

```
+-------------------------------------------------------------+---------+-------+
| NAME                                                        | ENABLED | TIMED |
+-------------------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/innobase_share_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/autoinc_persisted_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_flush_state_mutex          | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_LRU_list_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_free_list_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_free_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_hash_mutex             | NO      | NO    |
| wait/synch/mutex/innodb/buf_pool_zip_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/cache_last_read_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/dict_foreign_err_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/dict_persist_dirty_tables_mutex     | NO      | NO    |
| wait/synch/mutex/innodb/dict_sys_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/recalc_pool_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/fil_system_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/flush_list_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/fts_bg_threads_mutex                | NO      | NO    |
| wait/synch/mutex/innodb/fts_delete_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/fts_optimize_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/fts_doc_id_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/log_flush_order_mutex               | NO      | NO    |
| wait/synch/mutex/innodb/hash_table_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_bitmap_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_mutex                          | NO      | NO    |
| wait/synch/mutex/innodb/ibuf_pessimistic_insert_mutex       | NO      | NO    |
| wait/synch/mutex/innodb/log_sys_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/log_sys_write_mutex                 | NO      | NO    |
| wait/synch/mutex/innodb/mutex_list_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/page_zip_stat_per_index_mutex       | NO      | NO    |
| wait/synch/mutex/innodb/purge_sys_pq_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/recv_sys_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/recv_writer_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/redo_rseg_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/noredo_rseg_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_list_mutex                  | NO      | NO    |
| wait/synch/mutex/innodb/rw_lock_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/srv_dict_tmpfile_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/srv_innodb_monitor_mutex            | NO      | NO    |
| wait/synch/mutex/innodb/srv_misc_tmpfile_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/srv_monitor_file_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/buf_dblwr_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/trx_undo_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/trx_pool_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/trx_pool_manager_mutex              | NO      | NO    |
| wait/synch/mutex/innodb/srv_sys_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/lock_mutex                          | NO      | NO    |
| wait/synch/mutex/innodb/lock_wait_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/trx_mutex                           | NO      | NO    |
| wait/synch/mutex/innodb/srv_threads_mutex                   | NO      | NO    |
| wait/synch/mutex/innodb/rtr_active_mutex                    | NO      | NO    |
| wait/synch/mutex/innodb/rtr_match_mutex                     | NO      | NO    |
| wait/synch/mutex/innodb/rtr_path_mutex                      | NO      | NO    |
| wait/synch/mutex/innodb/rtr_ssn_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/trx_sys_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/zip_pad_mutex                       | NO      | NO    |
| wait/synch/mutex/innodb/master_key_id_mutex                 | 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.

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%';
+-----------------------------------------------------+---------+-------+
| NAME                                                | ENABLED | TIMED |
+-----------------------------------------------------+---------+-------+
| wait/synch/mutex/innodb/commit_cond_mutex           | YES     | YES   |
| wait/synch/mutex/innodb/innobase_share_mutex        | YES     | YES   |
| wait/synch/mutex/innodb/autoinc_mutex               | YES     | YES   |
...
| wait/synch/mutex/innodb/master_key_id_mutex         | YES     | YES   |
+-----------------------------------------------------+---------+-------+
49 rows in set (0.00 sec)
```

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'
       WHERE name like 'events_waits%';
Query OK, 3 rows affected (0.00 sec)
Rows matched: 3  Changed: 3  Warnings: 0
```

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 |
+--------------------------------+---------+
| events_stages_current          | NO      |
| events_stages_history          | NO      |
| events_stages_history_long     | NO      |
| events_statements_current      | YES     |
| events_statements_history      | YES     |
| events_statements_history_long | NO      |
| events_transactions_current    | YES     |
| events_transactions_history    | YES     |
| events_transactions_history_long | NO    |
| events_waits_current           | YES     |
| events_waits_history           | YES     |
| events_waits_history_long      | YES     |
| global_instrumentation         | YES     |
| thread_instrumentation         | YES     |
| statements_digest              | YES     |
+--------------------------------+---------+
15 rows in set (0.00 sec)
```

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 |
+-----------------------------------------------------------+------------+-------------------+
| wait/synch/mutex/innodb/trx_mutex                         |     201111 |           23.4719 |
| wait/synch/mutex/innodb/fil_system_mutex                  |      62244 |            9.6426 |
| wait/synch/mutex/innodb/redo_rseg_mutex                   |      48238 |            3.1135 |
| wait/synch/mutex/innodb/log_sys_mutex                     |      46113 |            2.0434 |
| wait/synch/mutex/innodb/trx_sys_mutex                     |      35134 |         1068.1588 |
| wait/synch/mutex/innodb/lock_mutex                        |      34872 |         1039.2589 |
| wait/synch/mutex/innodb/log_sys_write_mutex               |      17805 |         1526.0490 |
| wait/synch/mutex/innodb/dict_sys_mutex                    |      14912 |         1606.7348 |
| wait/synch/mutex/innodb/trx_undo_mutex                    |      10634 |            1.1424 |
| wait/synch/mutex/innodb/rw_lock_list_mutex                |       8538 |            0.1960 |
| wait/synch/mutex/innodb/buf_pool_free_list_mutex          |       5961 |            0.6473 |
| wait/synch/mutex/innodb/trx_pool_mutex                    |       4885 |         8821.7496 |
| wait/synch/mutex/innodb/buf_pool_LRU_list_mutex           |       4364 |            0.2077 |
| wait/synch/mutex/innodb/innobase_share_mutex              |       3212 |            0.2650 |
| wait/synch/mutex/innodb/flush_list_mutex                  |       3178 |            0.2349 |
| wait/synch/mutex/innodb/trx_pool_manager_mutex            |       2495 |            0.1310 |
| wait/synch/mutex/innodb/buf_pool_flush_state_mutex        |       1318 |            0.2161 |
| wait/synch/mutex/innodb/log_flush_order_mutex             |       1250 |            0.0893 |
| wait/synch/mutex/innodb/buf_dblwr_mutex                   |        951 |            0.0918 |
| wait/synch/mutex/innodb/recalc_pool_mutex                 |        670 |            0.0942 |
```

```
| 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` and `innodb_status_output_locks` 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` system variable to `ON`.

```
SET GLOBAL innodb_status_output=ON;
```

To disable the standard `InnoDB` Monitor, set `innodb_status_output` to `OFF`.

When you shut down the server, the `innodb_status_output` 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` 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` to `OFF`. Set `innodb_status_output` to `OFF` to also disable the `InnoDB` Standard Monitor.

When you shut down the server, the `innodb_status_output` and `innodb_status_output_locks` 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.

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      A;;
 2: len 7; hex 820000008b0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

------------------
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 6; hex 000000000202; asc       ;;
 1: len 6; hex 000000001e41; asc      A;;
 2: len 7; hex 820000008b0110; asc        ;;
 3: len 4; hex 80000001; asc     ;;

--------
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
```