MySQL Information Schema
Abstract

This is the MySQL Information Schema extract from the MySQL 8.0 Reference Manual.

For legal information, see the Legal Notices.

For help with using MySQL, please visit the MySQL Forums, where you can discuss your issues with other MySQL users.

Document generated on: 2020-03-25 (revision: 65457)
Table of Contents

Preface and Legal Notices ................................................................. v
1 INFORMATION_SCHEMA Tables ............................................................ 1
2 The INFORMATION_SCHEMA SCHEMATA Table ........................................... 3
3 The INFORMATION_SCHEMA TABLES Table ............................................... 5
4 The INFORMATION_SCHEMA COLUMNS Table .............................................. 9
5 The INFORMATION_SCHEMA STATISTICS Table ......................................... 13
6 The INFORMATION_SCHEMA USER_PRIVILEGES Table ............................... 17
7 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table ............................ 19
8 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table ............................. 21
9 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table .......................... 23
10 The INFORMATION_SCHEMA CHARACTER_SETS Table .............................. 25
11 The INFORMATION_SCHEMA COLLATIONS Table ....................................... 27
12 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table .... 29
13 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table .......................... 31
14 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table ............................ 33
15 The INFORMATION_SCHEMA ROUTINES Table ......................................... 35
16 The INFORMATION_SCHEMA VIEWS Table .............................................. 39
17 The INFORMATION_SCHEMA TRIGGERS Table .......................................... 41
18 The INFORMATION_SCHEMA PLUGINS Table .......................................... 45
19 The INFORMATION_SCHEMA ENGINES Table .......................................... 47
20 The INFORMATION_SCHEMA PARTITIONS Table ......................................... 49
21 The INFORMATION_SCHEMA EVENTS Table ............................................. 53
22 The INFORMATION_SCHEMA FILES Table .............................................. 57
23 The INFORMATION_SCHEMA PROCESSLIST Table ................................... 65
24 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table ................. 67
25 Extensions to SHOW Statements ......................................................... 69
26 MySQL 8.0 FAQ: INFORMATION_SCHEMA ............................................ 71
Preface and Legal Notices

This is the MySQL Information Schema extract from the MySQL 8.0 Reference Manual.

**Licensing information—MySQL 8.0.** This product may include third-party software, used under license. If you are using a Commercial release of MySQL 8.0, see the MySQL 8.0 Commercial Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Commercial release. If you are using a Community release of MySQL 8.0, see the MySQL 8.0 Community Release License Information User Manual for licensing information, including licensing information relating to third-party software that may be included in this Community release.

**Legal Notices**

Copyright © 1997, 2020, Oracle and/or its affiliates. All rights reserved.

This software and related documentation are provided under a license agreement containing restrictions on use and disclosure and are protected by intellectual property laws. Except as expressly permitted in your license agreement or allowed by law, you may not use, copy, reproduce, translate, broadcast, modify, license, transmit, distribute, exhibit, perform, publish, or display any part, in any form, or by any means. Reverse engineering, disassembly, or decompilation of this software, unless required by law for interoperability, is prohibited.

The information contained herein is subject to change without notice and is not warranted to be error-free. If you find any errors, please report them to us in writing.

If this is software or related documentation that is delivered to the U.S. Government or anyone licensing it on behalf of the U.S. Government, then the following notice is applicable:

U.S. GOVERNMENT END USERS: Oracle programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, delivered to U.S. Government end users are "commercial computer software" pursuant to the applicable Federal Acquisition Regulation and agency-specific supplemental regulations. As such, use, duplication, disclosure, modification, and adaptation of the programs, including any operating system, integrated software, any programs installed on the hardware, and/or documentation, shall be subject to license terms and license restrictions applicable to the programs. No other rights are granted to the U.S. Government.

This software or hardware is developed for general use in a variety of information management applications. It is not developed or intended for use in any inherently dangerous applications, including applications that may create a risk of personal injury. If you use this software or hardware in dangerous applications, then you shall be responsible to take all appropriate fail-safe, backup, redundancy, and other measures to ensure its safe use. Oracle Corporation and its affiliates disclaim any liability for any damages caused by use of this software or hardware in dangerous applications.

Oracle and Java are registered trademarks of Oracle and/or its affiliates. Other names may be trademarks of their respective owners.

Intel and Intel Xeon are trademarks or registered trademarks of Intel Corporation. All SPARC trademarks are used under license and are trademarks or registered trademarks of SPARC International, Inc. AMD, Opteron, the AMD logo, and the AMD Opteron logo are trademarks or registered trademarks of Advanced Micro Devices. UNIX is a registered trademark of The Open Group.

This software or hardware and documentation may provide access to or information about content, products, and services from third parties. Oracle Corporation and its affiliates are not responsible for and expressly disclaim all warranties of any kind with respect to third-party content, products, and services.
Documentation Accessibility

unless otherwise set forth in an applicable agreement between you and Oracle. Oracle Corporation and its affiliates will not be responsible for any loss, costs, or damages incurred due to your access to or use of third-party content, products, or services, except as set forth in an applicable agreement between you and Oracle.

This documentation is NOT distributed under a GPL license. Use of this documentation is subject to the following terms:

You may create a printed copy of this documentation solely for your own personal use. Conversion to other formats is allowed as long as the actual content is not altered or edited in any way. You shall not publish or distribute this documentation in any form or on any media, except if you distribute the documentation in a manner similar to how Oracle disseminates it (that is, electronically for download on a Web site with the software) or on a CD-ROM or similar medium, provided however that the documentation is disseminated together with the software on the same medium. Any other use, such as any dissemination of printed copies or use of this documentation, in whole or in part, in another publication, requires the prior written consent from an authorized representative of Oracle. Oracle and/or its affiliates reserve any and all rights to this documentation not expressly granted above.

Documentation Accessibility

For information about Oracle’s commitment to accessibility, visit the Oracle Accessibility Program website at
https://www.oracle.com/corporate/accessibility/.

Access to Oracle Support

Oracle customers that have purchased support have access to electronic support through My Oracle Support. For information, visit
https://www.oracle.com/corporate/accessibility/learning-support.html#support-tab.
Chapter 1 INFORMATION_SCHEMA Tables

`INFORMATION_SCHEMA` provides access to database *metadata*, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges. Other terms that are sometimes used for this information are *data dictionary* and *system catalog*.
Chapter 2 The INFORMATION_SCHEMA SCHEMATA Table

A schema is a database, so the SCHEMATA table provides information about databases.

The SCHEMATA table has these columns:

• CATALOG_NAME
  The name of the catalog to which the schema belongs. This value is always def.

• SCHEMA_NAME
  The name of the schema.

• DEFAULT_CHARACTER_SET_NAME
  The schema default character set.

• DEFAULT_COLLATION_NAME
  The schema default collation.

• SQL_PATH
  This value is always NULL.

• DEFAULT_ENCRYPTION
  The schema default encryption. This column was added in MySQL 8.0.16.

Schema names are also available from the SHOW DATABASES statement. See SHOW DATABASES Statement. The following statements are equivalent:

```sql
SELECT SCHEMA_NAME AS 'Database'
FROM INFORMATION_SCHEMA.SCHEMATA
[WHERE SCHEMA_NAME LIKE 'wild']
SHOW DATABASES
[LIKE 'wild']
```

You see only those databases for which you have some kind of privilege, unless you have the global SHOW DATABASES privilege.

Caution

Because any static global privilege is considered a privilege for all databases, any static global privilege enables a user to see all database names with SHOW DATABASES or by examining the SCHEMATA table of INFORMATION_SCHEMA, except databases that have been restricted at the database level by partial revokes.
Chapter 3 The INFORMATION_SCHEMA TABLES Table

The TABLES table provides information about tables in databases.

Columns in TABLES that represent table statistics hold cached values. The
information_schema_stats_expiry system variable defines the period of time before cached table
statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics
have expired, statistics are retrieved from storage engines when querying table statistics columns. To
update cached values at any time for a given table, use ANALYZE TABLE. To always retrieve the latest
statistics directly from storage engines, set information_schema_stats_expiry to 0. For more
information, see Optimizing INFORMATION_SCHEMA Queries.

Note
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.

The TABLES table has these columns:

- **TABLE_CATALOG**
  The name of the catalog to which the table belongs. This value is always def.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table belongs.

- **TABLE_NAME**
  The name of the table.

- **TABLE_TYPE**
  BASE TABLE for a table, VIEW for a view, or SYSTEM VIEW for an INFORMATION_SCHEMA table.

The TABLES table does not list TEMPORARY tables.

- **ENGINE**
  The storage engine for the table. See The InnoDB Storage Engine, and Alternative Storage Engines.
  For partitioned tables, ENGINE shows the name of the storage engine used by all partitions.

- **VERSION**
  This column is unused. With the removal of .frm files in MySQL 8.0, this column now reports a
  hardcoded value of 10, which is the last .frm file version used in MySQL 5.7.

- **ROW_FORMAT**
  The row-storage format (Fixed, Dynamic, Compressed, Redundant, Compact). For MyISAM tables,
  Dynamic corresponds to what myisamchk -dvv reports as Packed.

- **TABLE_ROWS**
The number of rows. Some storage engines, such as MyISAM, store the exact count. For other storage engines, such as InnoDB, this value is an approximation, and may vary from the actual value by as much as 40% to 50%. In such cases, use `SELECT COUNT(*)` to obtain an accurate count.

`TABLE_ROWS` is `NULL` for INFORMATION_SCHEMA tables.

For InnoDB tables, the row count is only a rough estimate used in SQL optimization. (This is also true if the InnoDB table is partitioned.)

- **AVG_ROW_LENGTH**
  The average row length.

- **DATA_LENGTH**
  For MyISAM, `DATA_LENGTH` is the length of the data file, in bytes.
  For InnoDB, `DATA_LENGTH` is the approximate amount of space allocated for the clustered index, in bytes. Specifically, it is the clustered index size, in pages, multiplied by the InnoDB page size.

Refer to the notes at the end of this section for information regarding other storage engines.

- **MAX_DATA_LENGTH**
  For MyISAM, `MAX_DATA_LENGTH` is maximum length of the data file. This is the total number of bytes of data that can be stored in the table, given the data pointer size used.

  Unused for InnoDB.

  Refer to the notes at the end of this section for information regarding other storage engines.

- **INDEX_LENGTH**
  For MyISAM, `INDEX_LENGTH` is the length of the index file, in bytes.
  For InnoDB, `INDEX_LENGTH` is the approximate amount of space allocated for non-clustered indexes, in bytes. Specifically, it is the sum of non-clustered index sizes, in pages, multiplied by the InnoDB page size.

  Refer to the notes at the end of this section for information regarding other storage engines.

- **DATA_FREE**
  The number of allocated but unused bytes.

  InnoDB tables report the free space of the tablespace to which the table belongs. For a table located in the shared tablespace, this is the free space of the shared tablespace. If you are using multiple tablespaces and the table has its own tablespace, the free space is for only that table. Free space means the number of bytes in completely free extents minus a safety margin. Even if free space displays as 0, it may be possible to insert rows as long as new extents need not be allocated.

  For NDB Cluster, `DATA_FREE` shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)

  For partitioned tables, this value is only an estimate and may not be absolutely correct. A more accurate method of obtaining this information in such cases is to query the INFORMATION_SCHEMA PARTITIONS table, as shown in this example:
SELECT SUM(DATA_FREE)
FROM INFORMATION_SCHEMA.PARTITIONS
WHERE TABLE_SCHEMA = 'mydb'
AND TABLE_NAME = 'mytable';

For more information, see Chapter 20, The INFORMATION_SCHEMA PARTITIONS Table.

- **AUTO_INCREMENT**
  
The next AUTO_INCREMENT value.

- **CREATE_TIME**
  
  When the table was created.

- **UPDATE_TIME**
  
  When the data file was last updated. For some storage engines, this value is NULL. For example, InnoDB stores multiple tables in its system tablespace and the data file timestamp does not apply. Even with file-per-table mode with each InnoDB table in a separate .ibd file, change buffering can delay the write to the data file, so the file modification time is different from the time of the last insert, update, or delete. For MyISAM, the data file timestamp is used; however, on Windows the timestamp is not updated by updates, so the value is inaccurate.

  UPDATE_TIME displays a timestamp value for the last UPDATE, INSERT, or DELETE performed on InnoDB tables that are not partitioned. For MVCC, the timestamp value reflects the COMMIT time, which is considered the last update time. Timestamps are not persisted when the server is restarted or when the table is evicted from the InnoDB data dictionary cache.

- **CHECK_TIME**
  
  When the table was last checked. Not all storage engines update this time, in which case, the value is always NULL.

  For partitioned InnoDB tables, CHECK_TIME is always NULL.

- **TABLE_COLLATION**
  
  The table default collation. The output does not explicitly list the table default character set, but the collation name begins with the character set name.

- **CHECKSUM**
  
  The live checksum value, if any.

- **CREATE_OPTIONS**
  
  Extra options used with CREATE TABLE. The original options specified when CREATE TABLE was executed are retained. The information reported may differ from current table settings and options.

  For InnoDB tables, the actual ROW_FORMAT and KEY_BLOCK_SIZE options are shown. Prior to MySQL 8.0, CREATE_OPTIONS shows the originally supplied ROW_FORMAT and KEY_BLOCK_SIZE. For more information, see CREATE TABLE Statement.

  CREATE_OPTIONS shows partitioned for a partitioned table. Prior to MySQL 8.0.16, it also shows the ENCRYPTION clause specified for tables created in file-per-table tablespaces. As of MySQL 8.0.16, it only shows the encryption clause for file-per-table tablespaces if the table is encrypted or if the specified encryption differs from the schema encryption. The encryption clause is not shown for tables...
Notes

created in general tablespaces. To identify encrypted file-per-table and general tablespaces, query the `INNODB_TABLESPACES_ENCRYPTION` column.

- **TABLE_COMMENT**

  The comment used when creating the table (or information as to why MySQL could not access the table information).

Notes

- For **NDB** tables, the output of this statement shows appropriate values for the `AVG_ROW_LENGTH` and `DATA_LENGTH` columns, with the exception that `BLOB` columns are not taken into account.

- For **NDB** tables, `DATA_LENGTH` includes data stored in main memory only; the `MAX_DATA_LENGTH` and `DATA_FREE` columns apply to Disk Data.

- For **NDB Cluster Disk Data tables**, `MAX_DATA_LENGTH` shows the space allocated for the disk part of a Disk Data table or fragment. (In-memory data resource usage is reported by the `DATA_LENGTH` column.)

- For **MEMORY** tables, the `DATA_LENGTH`, `MAX_DATA_LENGTH`, and `INDEX_LENGTH` values approximate the actual amount of allocated memory. The allocation algorithm reserves memory in large amounts to reduce the number of allocation operations.

- For views, most `TABLES` columns are 0 or `NULL` except that `TABLE_NAME` indicates the view name, `CREATE_TIME` indicates the creation time, and `TABLE_COMMENT` says `VIEW`.

Table information is also available from the `SHOW TABLE STATUS` and `SHOW TABLES` statements. See **SHOW TABLE STATUS Statement**, and **SHOW TABLES Statement**. The following statements are equivalent:

```sql
SELECT
  TABLE_NAME, ENGINE, VERSION, ROW_FORMAT, TABLE_ROWS, AVG_ROW_LENGTH,
  DATA_LENGTH, MAX_DATA_LENGTH, INDEX_LENGTH, DATA_FREE, AUTO_INCREMENT,
  CREATE_TIME, UPDATE_TIME, CHECK_TIME, TABLE_COLLATION, CHECKSUM,
  CREATE_OPTIONS, TABLE_COMMENT
FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
[AND table_name LIKE 'wild']
SHOW TABLE STATUS
FROM 'db_name'
[LIKE 'wild']
```

The following statements are equivalent:

```sql
SELECT
  TABLE_NAME, TABLE_TYPE
FROM INFORMATION_SCHEMA.TABLES
WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']
SHOW FULL TABLES
FROM 'db_name'
[LIKE 'wild']
```
Chapter 4 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables. The related ST_GEOMETRY_COLUMNS table provides information about table columns that store spatial data. See The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table.

The COLUMNS table has these columns:

- **TABLE_CATALOG**
  The name of the catalog to which the table containing the column belongs. This value is always def.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table containing the column belongs.

- **TABLE_NAME**
  The name of the table containing the column.

- **COLUMN_NAME**
  The name of the column.

- **ORDINAL_POSITION**
  The position of the column within the table. ORDINAL_POSITION is necessary because you might want to say ORDER BY ORDINAL_POSITION. Unlike SHOW COLUMNS, SELECT from the COLUMNS table does not have automatic ordering.

- **COLUMN_DEFAULT**
  The default value for the column. This is NULL if the column has an explicit default of NULL, or if the column definition includes no DEFAULT clause.

- **IS_NULLABLE**
  The column nullability. The value is YES if NULL values can be stored in the column, NO if not.

- **DATA_TYPE**
  The column data type.

  The DATA_TYPE value is the type name only with no other information. The COLUMN_TYPE value contains the type name and possibly other information such as the precision or length.

- **CHARACTER_MAXIMUM_LENGTH**
  For string columns, the maximum length in characters.

- **CHARACTER_OCTET_LENGTH**
  For string columns, the maximum length in bytes.

- **NUMERIC_PRECISION**
  For numeric columns, the numeric precision.

- **NUMERIC_SCALE**
For numeric columns, the numeric scale.

- **DATETIME_PRECISION**
  For temporal columns, the fractional seconds precision.

- **CHARACTER_SET_NAME**
  For character string columns, the character set name.

- **COLLATION_NAME**
  For character string columns, the collation name.

- **COLUMN_TYPE**
  The column data type.
  The **DATA_TYPE** value is the type name only with no other information. The **COLUMN_TYPE** value contains the type name and possibly other information such as the precision or length.

- **COLUMN_KEY**
  Whether the column is indexed:
  - If **COLUMN_KEY** is empty, the column either is not indexed or is indexed only as a secondary column in a multiple-column, nonunique index.
  - If **COLUMN_KEY** is **PRI**, the column is a **PRIMARY KEY** or is one of the columns in a multiple-column **PRIMARY KEY**.
  - If **COLUMN_KEY** is **UNI**, the column is the first column of a **UNIQUE** index. (A **UNIQUE** index permits multiple NULL values, but you can tell whether the column permits NULL by checking the Null column.)
  - If **COLUMN_KEY** is **MUL**, the column is the first column of a nonunique index in which multiple occurrences of a given value are permitted within the column.

  If more than one of the **COLUMN_KEY** values applies to a given column of a table, **COLUMN_KEY** displays the one with the highest priority, in the order **PRI**, **UNI**, **MUL**.

  A **UNIQUE** index may be displayed as **PRI** if it cannot contain NULL values and there is no **PRIMARY KEY** in the table. A **UNIQUE** index may display as **MUL** if several columns form a composite **UNIQUE** index; although the combination of the columns is unique, each column can still hold multiple occurrences of a given value.

- **EXTRA**
  Any additional information that is available about a given column. The value is nonempty in these cases:
  - **auto_increment** for columns that have the **AUTO_INCREMENT** attribute.
  - **on update CURRENT_TIMESTAMP** for **TIMESTAMP** or **DATETIME** columns that have the **ON UPDATE CURRENT_TIMESTAMP** attribute.
  - **STORED GENERATED** or **VIRTUAL GENERATED** for generated columns.
  - **DEFAULT_GENERATED** for columns that have an expression default value.
Notes

- **PRIVILEGES**
  The privileges you have for the column.

- **COLUMN_COMMENT**
  Any comment included in the column definition.

- **GENERATION_EXPRESSION**
  For generated columns, displays the expression used to compute column values. Empty for nongenerated columns. For information about generated columns, see CREATE TABLE and Generated Columns.

- **SRS_ID**
  This value applies to spatial columns. It contains the column SRID value that indicates the spatial reference system for values stored in the column. See Spatial Data Types, and Spatial Reference System Support. The value is NULL for nonspatial columns and spatial columns with no SRID attribute.

Notes

- In **SHOW COLUMNS**, the Type display includes values from several different COLUMNS columns.

- **CHARACTER_OCTET_LENGTH** should be the same as **CHARACTER_MAXIMUM_LENGTH**, except for multibyte character sets.

- **CHARACTER_SET_NAME** can be derived from COLLATION_NAME. For example, if you say **SHOW FULL COLUMNS FROM t**, and you see in the COLLATION_NAME column a value of utf8_swedish_ci, the character set is what is before the first underscore: utf8.

Column information is also available from the **SHOW COLUMNS** statement. See **SHOW COLUMNS Statement**. The following statements are nearly equivalent:

```sql
SELECT COLUMN_NAME, DATA_TYPE, IS_NULLABLE, COLUMN_DEFAULT
FROM INFORMATION_SCHEMA.COLUMNS
WHERE table_name = 'tbl_name'
  [AND table_schema = 'db_name']
  [AND column_name LIKE 'wild']
SHOW COLUMNS
FROM tbl_name
[FROM db_name]
[LIKE 'wild']
```
Chapter 5 The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

Columns in STATISTICS that represent table statistics hold cached values. The information_schema_stats_expiry system variable defines the period of time before cached table statistics expire. The default is 86400 seconds (24 hours). If there are no cached statistics or statistics have expired, statistics are retrieved from storage engines when querying table statistics columns. To update cached values at any time for a given table, use ANALYZE TABLE. To always retrieve the latest statistics directly from storage engines, set information_schema_stats_expiry=0. For more information, see Optimizing INFORMATION_SCHEMA Queries.

Note

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.

The STATISTICS table has these columns:

- **TABLE_CATALOG**
  The name of the catalog to which the table containing the index belongs. This value is always **def**.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table containing the index belongs.

- **TABLE_NAME**
  The name of the table containing the index.

- **NON_UNIQUE**
  0 if the index cannot contain duplicates, 1 if it can.

- **INDEX_SCHEMA**
  The name of the schema (database) to which the index belongs.

- **INDEX_NAME**
  The name of the index. If the index is the primary key, the name is always **PRIMARY**.

- **SEQ_IN_INDEX**
  The column sequence number in the index, starting with 1.

- **COLUMN_NAME**
  The column name. See also the description for the **EXPRESSION** column.

- **COLLATION**
  How the column is sorted in the index. This can have values **A** (ascending), **D** (descending), or **NULL** (not sorted).
• **CARDINALITY**

An estimate of the number of unique values in the index. To update this number, run `ANALYZE TABLE` or (for MyISAM tables) `myisamchk -a`.

CARDINALITY is counted based on statistics stored as integers, so the value is not necessarily exact even for small tables. The higher the cardinality, the greater the chance that MySQL uses the index when doing joins.

• **SUB_PART**

The index prefix. That is, the number of indexed characters if the column is only partly indexed, `NULL` if the entire column is indexed.

**Note**

Prefix *limits* are measured in bytes. However, prefix *lengths* for index specifications in `CREATE TABLE`, `ALTER TABLE`, and `CREATE INDEX` statements are interpreted as number of characters for nonbinary string types (`CHAR`, `VARCHAR`, `TEXT`) and number of bytes for binary string types (`BINARY`, `VARBINARY`, `BLOB`). Take this into account when specifying a prefix length for a nonbinary string column that uses a multibyte character set.

For additional information about index prefixes, see Column Indexes, and CREATE INDEX Statement.

• **PACKED**

Indicates how the key is packed. `NULL` if it is not.

• **NULLABLE**

Contains *YES* if the column may contain *NULL* values and '' if not.

• **INDEX_TYPE**

The index method used (`BTREE`, `FULLTEXT`, `HASH`, `RTREE`).

• **COMMENT**

Information about the index not described in its own column, such as *disabled* if the index is disabled.

• **INDEX_COMMENT**

Any comment provided for the index with a **COMMENT** attribute when the index was created.

• **IS_VISIBLE**

Whether the index is visible to the optimizer. See Invisible Indexes.

• **EXPRESSION**

MySQL 8.0.13 and higher supports functional key parts (see Functional Key Parts), which affects both the **COLUMN_NAME** and **EXPRESSION** columns:

• For a nonfunctional key part, **COLUMN_NAME** indicates the column indexed by the key part and **EXPRESSION** is `NULL`.

• For a functional key part, **COLUMN_NAME** column is `NULL` and **EXPRESSION** indicates the expression for the key part.
Notes

- There is no standard INFORMATION_SCHEMA table for indexes. The MySQL column list is similar to what SQL Server 2000 returns for `sp_statistics`, except that QUALIFIER and OWNER are replaced with CATALOG and SCHEMA, respectively.

Information about table indexes is also available from the SHOW INDEX statement. See SHOW INDEX Statement. The following statements are equivalent:

```sql
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'
SHOW INDEX
  FROM tbl_name
  FROM db_name
```
Chapter 6 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The USER_PRIVILEGES table provides information about global privileges. It takes its values from the mysql.user system table.

The USER_PRIVILEGES table has these columns:

- **GRANTEE**
  The name of the account to which the privilege is granted, in 'user_name'@'host_name' format.

- **TABLE_CATALOG**
  The name of the catalog. This value is always def.

- **PRIVILEGE_TYPE**
  The privilege granted. The value can be any privilege that can be granted at the global level; see GRANT Statement. Each row lists a single privilege, so there is one row per global privilege held by the grantee.

- **IS_GRANTABLE**
  YES if the user has the GRANT OPTION privilege, NO otherwise. The output does not list GRANT OPTION as a separate row with PRIVILEGE_TYPE='GRANT OPTION'.

**Notes**

- The USER_PRIVILEGES table is a nonstandard INFORMATION_SCHEMA table.

The following statements are not equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.USER_PRIVILEGES
SHOW GRANTS ...
```
Chapter 7 The INFORMATION_SCHEMA
SCHEMA_PRIVILEGES Table

The SCHEMA_PRIVILEGES table provides information about schema (database) privileges. It takes its values from the mysql.db system table.

The SCHEMA_PRIVILEGES table has these columns:

• GRANTEE
  The name of the account to which the privilege is granted, in 'user_name'@'host_name' format.

• TABLE_CATALOG
  The name of the catalog to which the schema belongs. This value is always def.

• TABLE_SCHEMA
  The name of the schema.

• PRIVILEGE_TYPE
  The privilege granted. The value can be any privilege that can be granted at the schema level; see GRANT Statement. Each row lists a single privilege, so there is one row per schema privilege held by the grantee.

• IS_GRANTABLE
  YES if the user has the GRANT OPTION privilege, NO otherwise. The output does not list GRANT OPTION as a separate row with PRIVILEGE_TYPE='GRANT OPTION'.

Notes

• The SCHEMA_PRIVILEGES table is a nonstandard INFORMATION_SCHEMA table.

The following statements are not equivalent:

SELECT ... FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES
SHOW GRANTS ...
Chapter 8 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. It takes its values from the mysql.tables_priv system table.

The TABLE_PRIVILEGES table has these columns:

- **GRANTEE**
  The name of the account to which the privilege is granted, in '{user_name}'@'{host_name}' format.

- **TABLE_CATALOG**
  The name of the catalog to which the table belongs. This value is always def.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table belongs.

- **TABLE_NAME**
  The name of the table.

- **PRIVILEGE_TYPE**
  The privilege granted. The value can be any privilege that can be granted at the table level; see GRANT Statement. Each row lists a single privilege, so there is one row per table privilege held by the grantee.

- **IS_GRANTABLE**
  YES if the user has the GRANT OPTION privilege, NO otherwise. The output does not list GRANT OPTION as a separate row with PRIVILEGE_TYPE='GRANT OPTION'.

Notes

- The TABLE_PRIVILEGES table is a nonstandard INFORMATION_SCHEMA table.

The following statements are not equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```
Chapter 9 The INFORMATION_SCHEMA
COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. It takes its values from the mysql.columns_priv system table.

The COLUMN_PRIVILEGES table has these columns:

• GRANTEE
  The name of the account to which the privilege is granted, in 'user_name'@'host_name' format.

• TABLE_CATALOG
  The name of the catalog to which the table containing the column belongs. This value is always def.

• TABLE_SCHEMA
  The name of the schema (database) to which the table containing the column belongs.

• TABLE_NAME
  The name of the table containing the column.

• COLUMN_NAME
  The name of the column.

• PRIVILEGE_TYPE
  The privilege granted. The value can be any privilege that can be granted at the column level; see GRANT Statement. Each row lists a single privilege, so there is one row per column privilege held by the grantee.

In the output from SHOW FULL COLUMNS, the privileges are all in one column and in lowercase, for example, select,insert,update,references. In COLUMN_PRIVILEGES, there is one privilege per row, in uppercase.

• IS_GRANTABLE
  YES if the user has the GRANT OPTION privilege, NO otherwise. The output does not list GRANT OPTION as a separate row with PRIVILEGE_TYPE='GRANT OPTION'.

Notes

• The COLUMN_PRIVILEGES table is a nonstandard INFORMATION_SCHEMA table.

The following statements are not equivalent:

SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
Chapter 10 The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

The CHARACTER_SETS table has these columns:

- **CHARACTER_SET_NAME**
  The character set name.

- **DEFAULT_COLLATE_NAME**
  The default collation for the character set.

- **DESCRIPTION**
  A description of the character set.

- **MAXLEN**
  The maximum number of bytes required to store one character.

Notes

Character set information is also available from the SHOW CHARACTER SET statement. See **SHOW CHARACTER SET Statement**. The following statements are equivalent:

```sql
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
[WHERE CHARACTER_SET_NAME LIKE 'wild']
SHOW CHARACTER SET
[LIKE 'wild']
```
Chapter 11 The INFORMATION_SCHEMA COLLATIONS Table

The **COLLATIONS** table provides information about collations for each character set.

The **COLLATIONS** table has these columns:

- **COLLATION_NAME**
  
  The collation name.

- **CHARACTER_SET_NAME**
  
  The name of the character set with which the collation is associated.

- **ID**
  
  The collation ID.

- **IS_DEFAULT**
  
  Whether the collation is the default for its character set.

- **IS_COMPILED**
  
  Whether the character set is compiled into the server.

- **SORTLEN**
  
  This is related to the amount of memory required to sort strings expressed in the character set.

- **PAD_ATTRIBUTE**
  
  The collation pad attribute.

Notes

Collation information is also available from the **SHOW COLLATION** statement. See **SHOW COLLATION Statement**. The following statements are equivalent:

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS  
  [WHERE COLLATION_NAME LIKE 'wild']
SHOW COLLATION  
  [LIKE 'wild']
```
Chapter 12 The INFORMATION_SCHEMA
COLLATION_CHARACTER_SET_APPLICABILITY Table

The COLLATION_CHARACTER_SET_APPLICABILITY table indicates what character set is applicable for what collation.

The COLLATION_CHARACTER_SET_APPLICABILITY table has these columns:

• COLLATION_NAME
  The collation name.

• CHARACTER_SET_NAME
  The name of the character set with which the collation is associated.

Notes

The COLLATION_CHARACTER_SET_APPLICABILITY columns are equivalent to the first two columns displayed by the SHOW COLLATION statement.
Chapter 13 The INFORMATION_SCHEMA
TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

The TABLE_CONSTRAINTS table has these columns:

• CONSTRAINT_CATALOG
  The name of the catalog to which the constraint belongs. This value is always def.

• CONSTRAINT_SCHEMA
  The name of the schema (database) to which the constraint belongs.

• TABLE_SCHEMA
  The name of the schema (database) to which the table belongs.

• TABLE_NAME
  The name of the table.

• CONSTRAINT_TYPE
  The type of constraint. The value can be UNIQUE, PRIMARY KEY, FOREIGN KEY, or (as of MySQL 8.0.16) CHECK. This is a CHAR (not ENUM) column.

  The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name column in the output from SHOW INDEX when the Non_unique column is 0.

• ENFORCED
  For CHECK constraints, the value is YES or NO to indicate whether the constraint is enforced. For other constraints, the value is always YES.

This column was added in MySQL 8.0.16.
Chapter 14 The INFORMATION_SCHEMA
KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints. This table provides no information about functional key parts because they are expressions and the table provides information only about columns.

The KEY_COLUMN_USAGE table has these columns:

- **CONSTRAINT_CATALOG**
  The name of the catalog to which the constraint belongs. This value is always def.

- **CONSTRAINT_SCHEMA**
  The name of the schema (database) to which the constraint belongs.

- **CONSTRAINT_NAME**
  The name of the constraint.

- **TABLE_CATALOG**
  The name of the catalog to which the table belongs. This value is always def.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table belongs.

- **TABLE_NAME**
  The name of the table that has the constraint.

- **COLUMN_NAME**
  The name of the column that has the constraint.

If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.

- **ORDINAL_POSITION**
  The column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.

- **POSITION_IN_UNIQUE_CONSTRAINT**
  NULL for unique and primary-key constraints. For foreign-key constraints, this column is the ordinal position in key of the table that is being referenced.

- **REFERENCED_TABLE_SCHEMA**
  The name of the schema referenced by the constraint.

- **REFERENCED_TABLE_NAME**
  The name of the table referenced by the constraint.
• **REFERENCED_COLUMN_NAME**

The name of the column referenced by the constraint.

Suppose that there are two tables named $t_1$ and $t_3$ that have the following definitions:

```sql
CREATE TABLE t1
{
  s1 INT,
  s2 INT,
  s3 INT,
  PRIMARY KEY(s3)
} ENGINE=InnoDB;
CREATE TABLE t3
{
  s1 INT,
  s2 INT,
  s3 INT,
  KEY(s1),
  CONSTRAINT CO FOREIGN KEY (s2) REFERENCES t1(s3)
} ENGINE=InnoDB;
```

For those two tables, the `KEY_COLUMN_USAGE` table has two rows:

- One row with **CONSTRAINT_NAME** = 'PRIMARY', **TABLE_NAME** = 't1', **COLUMN_NAME** = 's3', **ORDINAL_POSITION** = 1, **POSITION_IN_UNIQUE_CONSTRAINT** = NULL.

  For NDB: This value is always NULL.

- One row with **CONSTRAINT_NAME** = 'CO', **TABLE_NAME** = 't3', **COLUMN_NAME** = 's2', **ORDINAL_POSITION** = 1, **POSITION_IN_UNIQUE_CONSTRAINT** = 1.
Chapter 15 The INFORMATION_SCHEMA ROUTINES Table

The ROUTINES table provides information about stored routines (stored procedures and stored functions). The ROUTINES table does not include built-in SQL functions or user-defined functions (UDFs).

The ROUTINES table has these columns:

- **SPECIFIC_NAME**
  The name of the routine.

- **ROUTINE_CATALOG**
  The name of the catalog to which the routine belongs. This value is always `def`.

- **ROUTINE_SCHEMA**
  The name of the schema (database) to which the routine belongs.

- **ROUTINE_NAME**
  The name of the routine.

- **ROUTINE_TYPE**
  **PROCEDURE** for stored procedures, **FUNCTION** for stored functions.

- **DATA_TYPE**
  If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

  The **DATA_TYPE** value is the type name only with no other information. The **DTD_IDENTIFIER** value contains the type name and possibly other information such as the precision or length.

- **CHARACTER_MAXIMUM_LENGTH**
  For stored function string return values, the maximum length in characters. If the routine is a stored procedure, this value is **NULL**.

- **CHARACTER_OCTET_LENGTH**
  For stored function string return values, the maximum length in bytes. If the routine is a stored procedure, this value is **NULL**.

- **NUMERIC_PRECISION**
  For stored function numeric return values, the numeric precision. If the routine is a stored procedure, this value is **NULL**.

- **NUMERIC_SCALE**
  For stored function numeric return values, the numeric scale. If the routine is a stored procedure, this value is **NULL**.

- **DATETIME_PRECISION**
  For stored function temporal return values, the fractional seconds precision. If the routine is a stored procedure, this value is **NULL**.
• **CHARACTER_SET_NAME**

For stored function character string return values, the character set name. If the routine is a stored procedure, this value is **NULL**.

• **COLLATION_NAME**

For stored function character string return values, the collation name. If the routine is a stored procedure, this value is **NULL**.

• **DTD_IDENTIFIER**

If the routine is a stored function, the return value data type. If the routine is a stored procedure, this value is empty.

The **DATA_TYPE** value is the type name only with no other information. The **DTD_IDENTIFIER** value contains the type name and possibly other information such as the precision or length.

• **ROUTINE_BODY**

The language used for the routine definition. This value is always **SQL**.

• **ROUTINE_DEFINITION**

The text of the SQL statement executed by the routine.

• **EXTERNAL_NAME**

This value is always **NULL**.

• **EXTERNAL_LANGUAGE**

The language of the stored routine. The value is read from the `external_language` column of the `mysql.routines` data dictionary table.

• **PARAMETER_STYLE**

This value is always **SQL**.

• **IS_DETERMINISTIC**

**YES** or **NO**, depending on whether the routine is defined with the **DETERMINISTIC** characteristic.

• **SQL_DATA_ACCESS**

The data access characteristic for the routine. The value is one of **CONTAINS SQL, NO SQL, READS SQL DATA**, or **MODIFIES SQL DATA**.

• **SQL_PATH**

This value is always **NULL**.

• **SECURITY_TYPE**

The routine **SQL SECURITY** characteristic. The value is one of **DEFINER** or **INVOKER**.

• **CREATED**

The date and time when the routine was created. This is a **TIMESTAMP** value.
• **LAST_ALTERED**

The date and time when the routine was last modified. This is a `TIMESTAMP` value. If the routine has not been modified since its creation, this value is the same as the `CREATED` value.

• **SQL_MODE**

The SQL mode in effect when the routine was created or altered, and under which the routine executes. For the permitted values, see Server SQL Modes.

• **ROUTINE_COMMENT**

The text of the comment, if the routine has one. If not, this value is empty.

• **DEFINER**

The account named in the `DEFINER` clause (often the user who created the routine), in `user_name'@'host_name'` format.

• **CHARACTER_SET_CLIENT**

The session value of the `character_set_client` system variable when the routine was created.

• **COLLATION_CONNECTION**

The session value of the `collation_connection` system variable when the routine was created.

• **DATABASE_COLLATION**

The collation of the database with which the routine is associated.

Notes

• To see information about a routine, you must be the user named as the routine `DEFINER`, have the `SHOW_ROUTINE` privilege, have the `SELECT` privilege at the global level, or have the `CREATE ROUTINE`, `ALTER ROUTINE`, or `EXECUTE` privilege granted at a scope that includes the routine. The `ROUTINE_DEFINITION` column is `NULL` if you have only `CREATE ROUTINE`, `ALTER ROUTINE`, or `EXECUTE`.

• Information about stored function return values is also available in the `PARAMETERS` table. The return value row for a stored function can be identified as the row that has an `ORDINAL_POSITION` value of 0.
Chapter 16 The INFORMATION_SCHEMA VIEWS Table

The VIEWS table provides information about views in databases. You must have the SHOW VIEW privilege to access this table.

The VIEWS table has these columns:

- **TABLE_CATALOG**
  The name of the catalog to which the view belongs. This value is always `def`.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the view belongs.

- **TABLE_NAME**
  The name of the view.

- **VIEW_DEFINITION**
  The `SELECT` statement that provides the definition of the view. This column has most of what you see in the Create Table column that SHOW CREATE VIEW produces. Skip the words before `SELECT` and skip the words WITH CHECK OPTION. Suppose that the original statement was:

  ```sql
  CREATE VIEW v AS
  SELECT s2,s1 FROM t
  WHERE s1 > 5
  ORDER BY s1
  WITH CHECK OPTION;
  ```

  Then the view definition looks like this:

  ```sql
  SELECT s2,s1 FROM t WHERE s1 > 5 ORDER BY s1
  ```

- **CHECK_OPTION**
  The value of the CHECK_OPTION attribute. The value is one of NONE, CASCADE, or LOCAL.

- **IS_UPDATABLE**
  MySQL sets a flag, called the view updatability flag, at CREATE VIEW time. The flag is set to YES (true) if UPDATE and DELETE (and similar operations) are legal for the view. Otherwise, the flag is set to NO (false). The IS_UPDATABLE column in the VIEWS table displays the status of this flag. It means that the server always knows whether a view is updatable.

  If a view is not updatable, statements such UPDATE, DELETE, and INSERT are illegal and are rejected. (Even if a view is updatable, it might not be possible to insert into it; for details, refer to Updatable and Insertable Views.)

- **DEFINER**
  The account of the user who created the view, in 'user_name'@'host_name' format.

- **SECURITY_TYPE**
  The view SQL SECURITY characteristic. The value is one of DEFINER or INVOKER.

- **CHARACTER_SET_CLIENT**
The session value of the `character_set_client` system variable when the view was created.

- **COLLATION_CONNECTION**

  The session value of the `collation_connection` system variable when the view was created.

Notes

MySQL permits different `sql_mode` settings to tell the server the type of SQL syntax to support. For example, you might use the `ANSI` SQL mode to ensure MySQL correctly interprets the standard SQL concatenation operator, the double bar (||), in your queries. If you then create a view that concatenates items, you might worry that changing the `sql_mode` setting to a value different from `ANSI` could cause the view to become invalid. But this is not the case. No matter how you write out a view definition, MySQL always stores it the same way, in a canonical form. Here is an example that shows how the server changes a double bar concatenation operator to a `CONCAT()` function:

```
mysql> SET sql_mode = 'ANSI';
Query OK, 0 rows affected (0.00 sec)
mysql> CREATE VIEW test.v AS SELECT 'a' || 'b' as col1;
Query OK, 0 rows affected (0.00 sec)
mysql> SELECT VIEW_DEFINITION FROM INFORMATION_SCHEMA.VIEWS
    WHERE TABLE_SCHEMA = 'test' AND TABLE_NAME = 'v';
+----------------------------------+
| VIEW_DEFINITION                  |
+----------------------------------+
| select concat('a','b') AS `col1` |
+----------------------------------+
1 row in set (0.00 sec)
```

The advantage of storing a view definition in canonical form is that changes made later to the value of `sql_mode` do not affect the results from the view. However, an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.
Chapter 17 The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. To see information about a table's triggers, you must have the TRIGGER privilege for the table.

The TRIGGERS table has these columns:

- **TRIGGER_CATALOG**
  The name of the catalog to which the trigger belongs. This value is always def.

- **TRIGGER_SCHEMA**
  The name of the schema (database) to which the trigger belongs.

- **TRIGGER_NAME**
  The name of the trigger.

- **EVENT_MANIPULATION**
  The trigger event. This is the type of operation on the associated table for which the trigger activates. The value is INSERT (a row was inserted), DELETE (a row was deleted), or UPDATE (a row was modified).

- **EVENT_OBJECT_CATALOG**, **EVENT_OBJECT_SCHEMA**, and **EVENT_OBJECT_TABLE**
  As noted in Using Triggers, every trigger is associated with exactly one table. These columns indicate the catalog and schema (database) in which this table occurs, and the table name, respectively. The EVENT_OBJECT_CATALOG value is always def.

- **ACTION_ORDER**
  The ordinal position of the trigger's action within the list of triggers on the same table with the same EVENT_MANIPULATION and ACTION_TIMING values.

- **ACTION_CONDITION**
  This value is always NULL.

- **ACTION_STATEMENT**
  The trigger body; that is, the statement executed when the trigger activates. This text uses UTF-8 encoding.

- **ACTION_ORIENTATION**
  This value is always ROW.

- **ACTION_TIMING**
  Whether the trigger activates before or after the triggering event. The value is BEFORE or AFTER.

- **ACTION_REFERENCE_OLD_TABLE**
  This value is always NULL.

- **ACTION_REFERENCE_NEW_TABLE**
This value is always NULL.

- **ACTION_REFERENCE_OLD_ROW** and **ACTION_REFERENCE_NEW_ROW**

The old and new column identifiers, respectively. The **ACTION_REFERENCE_OLD_ROW** value is always **OLD** and the **ACTION_REFERENCE_NEW_ROW** value is always **NEW**.

- **CREATED**

The date and time when the trigger was created. This is a **TIMESTAMP(2)** value (with a fractional part in hundredths of seconds) for triggers.

- **SQL_MODE**

The SQL mode in effect when the trigger was created, and under which the trigger executes. For the permitted values, see Server SQL Modes.

- **DEFINER**

The account named in the **DEFINER** clause (often the user who created the trigger), in `'user_name'@'host_name'` format.

- **CHARACTER_SET_CLIENT**

The session value of the **character_set_client** system variable when the trigger was created.

- **COLLATION_CONNECTION**

The session value of the **collation_connection** system variable when the trigger was created.

- **DATABASE_COLLATION**

The collation of the database with which the trigger is associated.

### Example

The following example uses the **ins_sum** trigger defined in Using Triggers:

```sql
mysql> SELECT * FROM INFORMATION_SCHEMA.TRIGGERS
WHERE TRIGGER_SCHEMA='test' AND TRIGGER_NAME='ins_sum'\G
*************************** 1. row ***************************
TRIGGER_CATALOG: def
TRIGGER_SCHEMA: test
TRIGGER_NAME: ins_sum
EVENT_MANIPULATION: INSERT
EVENT_OBJECT_CATALOG: def
EVENT_OBJECT_SCHEMA: test
EVENT_OBJECT_TABLE: account
ACTION_ORDER: 1
ACTION_CONDITION: NULL
ACTION_STATEMENT: SET @sum = @sum + NEW.amount
ACTION_ORIENTATION: ROW
ACTION_TIMING: BEFORE
ACTION_REFERENCE_OLD_TABLE: NULL
ACTION_REFERENCE_NEW_TABLE: NULL
ACTION_REFERENCE_OLD_ROW: OLD
ACTION_REFERENCE_NEW_ROW: NEW
CREATED: 2018-08-08 10:10:12.61
SQL_MODE: ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,
NO_ZERO_IN_DATE,NO_ZERO_DATE,
Trigger information is also available from the `SHOW TRIGGERS` statement. See `SHOW TRIGGERS Statement`.
Chapter 18 The INFORMATION_SCHEMA PLUGINS Table

The PLUGINS table provides information about server plugins.

The PLUGINS table has these columns:

• **PLUGIN_NAME**
  The name used to refer to the plugin in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`.

• **PLUGIN_VERSION**
  The version from the plugin's general type descriptor.

• **PLUGIN_STATUS**
  The plugin status, one of `ACTIVE`, `INACTIVE`, `DISABLED`, `DELETING`, or `DELETED`.

• **PLUGIN_TYPE**
  The type of plugin, such as `STORAGE ENGINE`, `INFORMATION_SCHEMA`, or `AUTHENTICATION`.

• **PLUGIN_TYPE_VERSION**
  The version from the plugin's type-specific descriptor.

• **PLUGIN_LIBRARY**
  The name of the plugin shared library file. This is the name used to refer to the plugin file in statements such as `INSTALL PLUGIN` and `UNINSTALL PLUGIN`. This file is located in the directory named by the `plugin_dir` system variable. If the library name is `NULL`, the plugin is compiled in and cannot be uninstalled with `UNINSTALL PLUGIN`.

• **PLUGIN_LIBRARY_VERSION**
  The plugin API interface version.

• **PLUGIN_AUTHOR**
  The plugin author.

• **PLUGIN_DESCRIPTION**
  A short description of the plugin.

• **PLUGIN_LICENSE**
  How the plugin is licensed (for example, `GPL`).

• **LOAD_OPTION**
  How the plugin was loaded. The value is `OFF`, `ON`, `FORCE`, or `FORCE_PLUS_Permanent`. See Installing and Uninstalling Plugins.

Notes

• The PLUGINS table is a nonstandard INFORMATION_SCHEMA table.
• For plugins installed with `INSTALL PLUGIN`, the `PLUGIN_NAME` and `PLUGIN_LIBRARY` values are also registered in the `mysql.plugin` table.

• For information about plugin data structures that form the basis of the information in the `PLUGINS` table, see The MySQL Plugin API.

Plugin information is also available from the `SHOW PLUGINS` statement. See SHOW PLUGINS Statement. These statements are equivalent:

```sql
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;
SHOW PLUGINS;
```
Chapter 19 The INFORMATION_SCHEMA ENGINES Table

The ENGINES table provides information about storage engines. This is particularly useful for checking whether a storage engine is supported, or to see what the default engine is.

The ENGINES table has these columns:

• ENGINE
  The name of the storage engine.

• SUPPORT
  The server's level of support for the storage engine, as shown in the following table.

<table>
<thead>
<tr>
<th>Value</th>
<th>Meaning</th>
</tr>
</thead>
<tbody>
<tr>
<td>YES</td>
<td>The engine is supported and is active</td>
</tr>
<tr>
<td>DEFAULT</td>
<td>Like YES, plus this is the default engine</td>
</tr>
<tr>
<td>NO</td>
<td>The engine is not supported</td>
</tr>
<tr>
<td>DISABLED</td>
<td>The engine is supported but has been disabled</td>
</tr>
</tbody>
</table>

A value of NO means that the server was compiled without support for the engine, so it cannot be enabled at runtime.

A value of DISABLED occurs either because the server was started with an option that disables the engine, or because not all options required to enable it were given. In the latter case, the error log should contain a reason indicating why the option is disabled. See The Error Log.

You might also see DISABLED for a storage engine if the server was compiled to support it, but was started with a --skip-engine_name option. For the NDB storage engine, DISABLED means the server was compiled with support for NDB Cluster, but was not started with the --ndbcluster option.

All MySQL servers support MyISAM tables. It is not possible to disable MyISAM.

• COMMENT
  A brief description of the storage engine.

• TRANSACTIONS
  Whether the storage engine supports transactions.

• XA
  Whether the storage engine supports XA transactions.

• SAVEPOINTS
  Whether the storage engine supports savepoints.

Notes

• The ENGINES table is a nonstandard INFORMATION_SCHEMA table.

Storage engine information is also available from the SHOW ENGINES statement. See SHOW ENGINES Statement. The following statements are equivalent:
<table>
<thead>
<tr>
<th>SQL Command</th>
</tr>
</thead>
<tbody>
<tr>
<td>SELECT * FROM INFORMATION_SCHEMA.ENGINES</td>
</tr>
<tr>
<td>SHOW ENGINES</td>
</tr>
</tbody>
</table>
Chapter 20 The INFORMATION_SCHEMA PARTITIONS Table

The PARTITIONS table provides information about table partitions. Each row in this table corresponds to an individual partition or subpartition of a partitioned table. For more information about partitioning tables, see Partitioning.

The PARTITIONS table has these columns:

- **TABLE_CATALOG**
  The name of the catalog to which the table belongs. This value is always `def`.

- **TABLE_SCHEMA**
  The name of the schema (database) to which the table belongs.

- **TABLE_NAME**
  The name of the table containing the partition.

- **PARTITION_NAME**
  The name of the partition.

- **SUBPARTITION_NAME**
  If the PARTITIONS table row represents a subpartition, the name of subpartition; otherwise `NULL`. For NDB: This value is always `NULL`.

- **PARTITION_ORDINAL_POSITION**
  All partitions are indexed in the same order as they are defined, with 1 being the number assigned to the first partition. The indexing can change as partitions are added, dropped, and reorganized; the number shown is this column reflects the current order, taking into account any indexing changes.

- **SUBPARTITION_ORDINAL_POSITION**
  Subpartitions within a given partition are also indexed and reindexed in the same manner as partitions are indexed within a table.

- **PARTITION_METHOD**
  One of the values `RANGE`, `LIST`, `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available partitioning types as discussed in Partitioning Types.

- **SUBPARTITION_METHOD**
  One of the values `HASH`, `LINEAR HASH`, `KEY`, or `LINEAR KEY`; that is, one of the available subpartitioning types as discussed in Subpartitioning.

- **PARTITION_EXPRESSION**
  The expression for the partitioning function used in the `CREATE TABLE` or `ALTER TABLE` statement that created the table's current partitioning scheme.

For example, consider a partitioned table created in the `test` database using this statement:

```sql
CREATE TABLE tp (
```
The **PARTITION_EXPRESSION** column in a **PARTITIONS** table row for a partition from this table displays **c1 + c2**, as shown here:

```
mysql> SELECT DISTINCT PARTITION_EXPRESSION
           FROM INFORMATION_SCHEMA.PARTITIONS
           WHERE TABLE_NAME='tp' AND TABLE_SCHEMA='test';
+----------------------+
| PARTITION_EXPRESSION |
+----------------------+
| c1 + c2              |
+----------------------+
```

**SUBPARTITION_EXPRESSION**

This works in the same fashion for the subpartitioning expression that defines the subpartitioning for a table as **PARTITION_EXPRESSION** does for the partitioning expression used to define a table's partitioning.

If the table has no subpartitions, this column is **NULL**.

**PARTITION_DESCRIPTION**

This column is used for **RANGE** and **LIST** partitions. For a **RANGE** partition, it contains the value set in the partition's **VALUES LESS THAN** clause, which can be either an integer or **MAXVALUE**. For a **LIST** partition, this column contains the values defined in the partition's **VALUES IN** clause, which is a list of comma-separated integer values.

For partitions whose **PARTITION_METHOD** is other than **RANGE** or **LIST**, this column is always **NULL**.

**TABLE_ROWS**

The number of table rows in the partition.

For partitioned **InnoDB** tables, the row count given in the **TABLE_ROWS** column is only an estimated value used in SQL optimization, and may not always be exact.

For **NDB** tables, you can also obtain this information using the **ndb_desc** utility.

**AVG_ROW_LENGTH**

The average length of the rows stored in this partition or subpartition, in bytes. This is the same as **DATA_LENGTH** divided by **TABLE_ROWS**.

For **NDB** tables, you can also obtain this information using the **ndb_desc** utility.

**DATA_LENGTH**

The total length of all rows stored in this partition or subpartition, in bytes; that is, the total number of bytes stored in the partition or subpartition.

For **NDB** tables, you can also obtain this information using the **ndb_desc** utility.

**MAX_DATA_LENGTH**
The maximum number of bytes that can be stored in this partition or subpartition. For NDB tables, you can also obtain this information using the ndb_desc utility.

• **INDEX_LENGTH**
  The length of the index file for this partition or subpartition, in bytes.
  For partitions of NDB tables, whether the tables use implicit or explicit partitioning, the INDEX_LENGTH column value is always 0. However, you can obtain equivalent information using the ndb_desc utility.

• **DATA_FREE**
  The number of bytes allocated to the partition or subpartition but not used.
  For NDB tables, you can also obtain this information using the ndb_desc utility.

• **CREATE_TIME**
  The time that the partition or subpartition was created.

• **UPDATE_TIME**
  The time that the partition or subpartition was last modified.

• **CHECK_TIME**
  The last time that the table to which this partition or subpartition belongs was checked.
  For partitioned InnoDB tables, the value is always NULL.

• **CHECKSUM**
  The checksum value, if any; otherwise NULL.

• **PARTITION_COMMENT**
  The text of the comment, if the partition has one. If not, this value is empty.
  The maximum length for a partition comment is defined as 1024 characters, and the display width of the PARTITION_COMMENT column is also 1024, characters to match this limit.

• **NODEGROUP**
  This is the nodegroup to which the partition belongs. This is relevant only to NDB Cluster tables; otherwise, the value is always 0.

• **TABLESPACE_NAME**
  The name of the tablespace to which the partition belongs. The value is always DEFAULT, unless the table uses the NDB storage engine (see the Notes at the end of this section).

**Notes**

• The PARTITIONS table is a nonstandard INFORMATION_SCHEMA table.

• A table using any storage engine other than NDB and which is not partitioned has one row in the PARTITIONS table. However, the values of the PARTITION_NAME, SUBPARTITION_NAME,
Notes

PARTITION_ORDINAL_POSITION, SUBPARTITION_ORDINAL_POSITION, PARTITION_METHOD,
SUBPARTITION_METHOD, PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, and
PARTITION_DESCRIPTION columns are all NULL. Also, the PARTITION_COMMENT column in this case
is blank.

• An NDB table which is not explicitly partitioned has one row in the PARTITIONS table for each data node
in the NDB cluster. For each such row:
  
  • The SUBPARTITION_NAME, SUBPARTITION_ORDINAL_POSITION, SUBPARTITION_METHOD,
    PARTITION_EXPRESSION, SUBPARTITION_EXPRESSION, CREATE_TIME, UPDATE_TIME,
    CHECK_TIME, CHECKSUM, and TABLESPACE_NAME columns are all NULL.
  
  • The PARTITION_METHOD is always AUTO.
  
  • The NODEGROUP column is default.
  
  • The PARTITION_EXPRESSION and PARTITION_COMMENT columns are empty.
Chapter 21 The INFORMATION_SCHEMA EVENTS Table

The EVENTS table provides information about Event Manager events, which are discussed in Using the Event Scheduler.

The EVENTS table has these columns:

- **EVENT_CATALOG**
  The name of the catalog to which the event belongs. This value is always def.

- **EVENT_SCHEMA**
  The name of the schema (database) to which the event belongs.

- **EVENT_NAME**
  The name of the event.

- **DEFINER**
  The account named in the DEFINER clause (often the user who created the event), in 'user_name'@'host_name' format.

- **TIME_ZONE**
  The event time zone, which is the time zone used for scheduling the event and that is in effect within the event as it executes. The default value is SYSTEM.

- **EVENT_BODY**
  The language used for the statements in the event's DO clause. The value is always SQL.

- **EVENT_DEFINITION**
  The text of the SQL statement making up the event's DO clause; in other words, the statement executed by this event.

- **EVENT_TYPE**
  The event repetition type, either ONE_TIME (transient) or RECURRING (repeating).

- **EXECUTE_AT**
  For a one-time event, this is the DATETIME value specified in the AT clause of the CREATE EVENT statement used to create the event, or of the last ALTER EVENT statement that modified the event. The value shown in this column reflects the addition or subtraction of any INTERVAL value included in the event's AT clause. For example, if an event is created using ON SCHEDULE AT CURRENT_TIMESTAMP + '1:6' DAY_HOUR, and the event was created at 2018-02-09 14:05:30, the value shown in this column would be '2018-02-10 20:05:30'. If the event's timing is determined by an EVERY clause instead of an AT clause (that is, if the event is recurring), the value of this column is NULL.

- **INTERVAL_VALUE**
  For a recurring event, the number of intervals to wait between event executions. For a transient event, the value is always NULL.

- **INTERVAL_FIELD**
The time units used for the interval which a recurring event waits before repeating. For a transient event, the value is always NULL.

- **SQL_MODE**

The SQL mode in effect when the event was created or altered, and under which the event executes. For the permitted values, see Server SQL Modes.

- **STARTS**

The start date and time for a recurring event. This is displayed as a DATETIME value, and is NULL if no start date and time are defined for the event. For a transient event, this column is always NULL. For a recurring event whose definition includes a STARTS clause, this column contains the corresponding DATETIME value. As with the EXECUTE_AT column, this value resolves any expressions used. If there is no STARTS clause affecting the timing of the event, this column is NULL.

- **ENDS**

For a recurring event whose definition includes a ENDS clause, this column contains the corresponding DATETIME value. As with the EXECUTE_AT column, this value resolves any expressions used. If there is no ENDS clause affecting the timing of the event, this column is NULL.

- **STATUS**

The event status. One of ENABLED, DISABLED, or SLAVESIDE_DISABLED. SLAVESIDE_DISABLED indicates that the creation of the event occurred on another MySQL server acting as a replication master and replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. For more information, see Replication of Invoked Features.

- **ON_COMPLETION**

One of the two values PRESERVE or NOT PRESERVE.

- **CREATED**

The date and time when the event was created. This is a TIMESTAMP value.

- **LAST_ALTERED**

The date and time when the event was last modified. This is a TIMESTAMP value. If the event has not been modified since its creation, this value is the same as the CREATED value.

- **LAST_EXECUTED**

The date and time when the event last executed. This is a DATETIME value. If the event has never executed, this column is NULL.

LAST_EXECUTED indicates when the event started. As a result, the ENDS column is never less than LAST_EXECUTED.

- **EVENT_COMMENT**

The text of the comment, if the event has one. If not, this value is empty.
• **ORIGINATOR**
  The server ID of the MySQL server on which the event was created; used in replication. This value may be updated by `ALTER EVENT` to the server ID of the server on which that statement occurs, if executed on a master server. The default value is 0.

• **CHARACTER_SET_CLIENT**
  The session value of the `character_set_client` system variable when the event was created.

• **COLLATION_CONNECTION**
  The session value of the `collation_connection` system variable when the event was created.

• **DATABASE_COLLATION**
  The collation of the database with which the event is associated.

**Notes**

• The `EVENTS` table is a nonstandard `INFORMATION_SCHEMA` table.

• Times in the `EVENTS` table are displayed using the event time zone, the current session time zone, or UTC, as described in Event Metadata.

• For more information about `SLAVESIDE_DISABLED` and the `ORIGINATOR` column, see Replication of Invoked Features.

**Example**

Suppose that the user 'jon'@'ghidora' creates an event named `e_daily`, and then modifies it a few minutes later using an `ALTER EVENT` statement, as shown here:

```sql
DELIMITER |
CREATE EVENT e_daily
ON SCHEDULE
  EVERY 1 DAY
COMMENT 'Saves total number of sessions then clears the table each day'
DO
BEGIN
  INSERT INTO site_activity.totals (time, total)
  SELECT CURRENT_TIMESTAMP, COUNT(*)
  FROM site_activity.sessions;
  DELETE FROM site_activity.sessions;
END |
DELIMITER ;
ALTER EVENT e_daily
ENABLE;
```

(Note that comments can span multiple lines.)

This user can then run the following `SELECT` statement, and obtain the output shown:

```sql
mysql> SELECT * FROM INFORMATION_SCHEMA.EVENTS
WHERE EVENT_NAME = 'e_daily'
AND EVENT_SCHEMA = 'myschema'\G
*************************** 1. row ***************************
EVENT_CATALOG: def
EVENT_SCHEMA: myschema
EVENT_NAME: e_daily
DEFINER: jon@ghidora
```
TIME_ZONE: SYSTEM
EVENT_BODY: SQL
EVENT_DEFINITION: BEGIN
    INSERT INTO site_activity.totals (time, total)
    SELECT CURRENT_TIMESTAMP, COUNT(*)
    FROM site_activity.sessions;
    DELETE FROM site_activity.sessions;
END
EVENT_TYPE: RECURRING
EXECUTE_AT: NULL
INTERVAL_VALUE: 1
INTERVAL_FIELD: DAY
SQL_MODE: ONLY_FULL_GROUP_BY, STRICT_TRANS_TABLES,
    NO_ZERO_IN_DATE, NO_ZERO_DATE,
    ERROR_FOR_DIVISION_BY_ZERO,
    NO_ENGINE_SUBSTITUTION
STARTS: 2018-08-08 11:06:34
ENDS: NULL
STATUS: ENABLED
ON_COMPLETION: NOT PRESERVE
CREATED: 2018-08-08 11:06:34
LAST_ALTERED: 2018-08-08 11:06:34
LAST_EXECUTED: 2018-08-08 16:06:34
EVENT_COMMENT: Saves total number of sessions then clears the table each day
ORIGINATOR: 1
CHARACTER_SET_CLIENT: utf8mb4
COLLATION_CONNECTION: utf8mb4_0900_ai_ci
DATABASE_COLLATION: utf8mb4_0900_ai_ci

Event information is also available from the SHOW EVENTS statement. See SHOW EVENTS Statement. The following statements are equivalent:

SELECT
    EVENT_SCHEMA, EVENT_NAME, DEFINER, TIME_ZONE, EVENT_TYPE, EXECUTE_AT,
    INTERVAL_VALUE, INTERVAL_FIELD, STARTS, ENDS, STATUS, ORIGINATOR,
    CHARACTER_SET_CLIENT, COLLATION_CONNECTION, DATABASE_COLLATION
FROM INFORMATION_SCHEMA.EVENTS
WHERE table_schema = 'db_name' [AND column_name LIKE 'wild']
SHOW EVENTS [FROM db_name] [LIKE 'wild']
Chapter 22 The INFORMATION_SCHEMA FILES Table

The **FILES** table provides information about the files in which MySQL tablespace data is stored.

The **FILES** table provides information about InnoDB data files. In NDB Cluster, this table also provides information about the files in which NDB Cluster Disk Data tables are stored. For additional information specific to InnoDB, see **InnoDB Notes**, later in this section; for additional information specific to NDB Cluster, see **NDB Notes**.

The **FILES** table has these columns:

- **FILE_ID**
  
  For **InnoDB**: The tablespace ID, also referred to as the `space_id` or `fil_space_t::id`.
  
  For **NDB**: A file identifier. **FILE_ID** column values are auto-generated.

- **FILE_NAME**
  
  For **InnoDB**: The name of the data file. File-per-table and general tablespaces have an `.ibd` file name extension. Undo tablespaces are prefixed by `undo`. The system tablespace is prefixed by `ibdata`. The global temporary tablespace is prefixed by `ibtmp`. The file name includes the file path, which may be relative to the MySQL data directory (the value of the `datadir` system variable).
  
  For **NDB**: The name of an UNDO log file created by `CREATE LOGFILE GROUP` or `ALTER LOGFILE GROUP`, or of a data file created by `CREATE TABLESPACE` or `ALTER TABLESPACE`.

- **FILE_TYPE**
  
  For **InnoDB**: The tablespace file type. There are three possible file types for InnoDB files. `TABLESPACE` is the file type for any system, general, or file-per-table tablespace file that holds tables, indexes, or other forms of user data. `TEMPORARY` is the file type for temporary tablespaces. `UNDO LOG` is the file type for undo tablespaces, which hold undo records.
  
  For **NDB**: One of the values `UNDO LOG`, `DATAFILE`, or `TABLESPACE`.

- **TABLESPACE_NAME**
  
  For **InnoDB**: The SQL name for the tablespace. A general tablespace name is the `SYS_TABLESPACES.NAME` value. For other tablespace files, names start with `innodb_`, such as `innodb_system`, `innodb_undo`, and `innodb_file_per_table`. The file-per-table tablespace name format is `innodb_file_per_table_##`, where `##` is the tablespace ID.
  
  For **NDB**: The name of the tablespace with which the file is associated.

- **TABLE_CATALOG**
  
  This value is always empty.

- **TABLE_SCHEMA**
  
  This value is always **NULL**.

- **TABLE_NAME**
  
  For **InnoDB**: This value is always **NULL**.
  
  For **NDB**: The name of the Disk Data table with which the file is associated, if any.
• **LOGFILE_GROUP_NAME**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: The name of the log file group to which the log file or data file belongs.

• **LOGFILE_GROUP_NUMBER**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For an **UNDO** log file, the auto-generated ID number of the log file group to which the log file belongs.

• **ENGINE**
  For **InnoDB**: This value is always **InnoDB**.
  For **NDB**: For an NDB Cluster Disk Data log file or data file, this value is always **NDB** or **NDBCLUSTER**.

• **FULLTEXT_KEYS**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For an NDB Cluster Disk Data log file or data file, this value is always empty.

• **DELETED_ROWS**
  For **InnoDB**: This value is always **NULL**.

• **UPDATE_COUNT**
  For **InnoDB**: This value is always **NULL**.

• **FREE_EXTENTS**
  For **InnoDB**: The number of fully free extents in the current data file.
  For **NDB**: The number of extents which have not yet been used by the file.

• **TOTAL_EXTENTS**
  For **InnoDB**: The number of full extents used in the current data file. Any partial extent at the end of the file is not counted.
  For **NDB**: The total number of extents allocated to the file.

• **EXTENT_SIZE**
  For **InnoDB**: Extent size is 1048576 (1MB) for files with a 4KB, 8KB, or 16KB page size. Extent size is 2097152 bytes (2MB) for files with a 32KB page size, and 4194304 (4MB) for files with a 64KB page size. **FILES** does not report **InnoDB** page size. Page size is defined by the **innodb_page_size** system variable. Extent size information can also be retrieved from the **INNODB_TABLESPACES** table where **FILES.FILE_ID = INNODB_TABLESPACES.SPACE**.
  For **NDB**: The size of an extent for the file in bytes.

• **INITIAL_SIZE**
  For **InnoDB**: The initial size of the file in bytes.
For **NDB**: The size of the file in bytes. This is the same value that was used in the `INITIAL_SIZE` clause of the `CREATE LOGFILE GROUP`, `ALTER LOGFILE GROUP`, `CREATE TABLESPACE`, or `ALTER TABLESPACE` statement used to create the file.

- **MAXIMUM_SIZE**

  For **InnoDB**: The maximum number of bytes permitted in the file. The value is `NULL` for all data files except for predefined system tablespace data files. Maximum system tablespace file size is defined by `innodb_data_file_path`. Maximum global temporary tablespace file size is defined by `innodb_temp_data_file_path`. A `NULL` value for a predefined system tablespace data file indicates that a file size limit was not defined explicitly.

  For **NDB**: For NDB Cluster Disk Data files, this value is always the same as the `INITIAL_SIZE` value.

- **AUTOEXTEND_SIZE**

  For **InnoDB**: `AUTOEXTEND_SIZE` is the auto-extend size defined by `innodb_data_file_path` for the system tablespace, or by `innodb_temp_data_file_path` for the global temporary tablespace.

  For **NDB**: For NDB Cluster Disk Data files, this value is always empty.

- **CREATION_TIME**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: The date and time when the file was created.

- **LAST_UPDATE_TIME**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: The date and time when the file was last modified.

- **LAST_ACCESS_TIME**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: The date and time when the file was last accessed by the server.

- **RECOVER_TIME**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: For NDB Cluster Disk Data files, this value is always `0`.

- **TRANSACTION_COUNTER**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: For NDB Cluster Disk Data files, this value is always `0`.

- **VERSION**

  For **InnoDB**: This value is always `NULL`.

  For **NDB**: For NDB Cluster Disk Data files, this value is always `NULL`.

- **ROW_FORMAT**
For **InnoDB**: This value is always **NULL**.
For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **TABLE_ROWS**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **AVG_ROW_LENGTH**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **DATA_LENGTH**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **MAX_DATA_LENGTH**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **INDEX_LENGTH**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **DATA_FREE**
  For **InnoDB**: The total amount of free space (in bytes) for the entire tablespace. Predefined system tablespaces, which include the system tablespace and temporary table tablespaces, may have one or more data files.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **CREATE_TIME**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **UPDATE_TIME**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **CHECK_TIME**
  For **InnoDB**: This value is always **NULL**.
  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.
Notes

- **CHECKSUM**

  For **InnoDB**: This value is always **NULL**.

  For **NDB**: For NDB Cluster Disk Data files, this value is always **NULL**.

- **STATUS**

  For **InnoDB**: This value is **NORMAL** by default. **InnoDB** file-per-table tablespaces may report **IMPORTING**, which indicates that the tablespace is not yet available.

  For **NDB**: For NDB Cluster Disk Data files, this value is always **NORMAL**.

- **EXTRA**

  For **InnoDB**: This value is always **NULL**.

  For **NDB**: For NDB Cluster Disk Data files, the **EXTRA** column shows which data node the file belongs to (each data node having its own copy), as well as the size of its undo buffer. Suppose that you use this statement on an NDB Cluster with four data nodes:

  ```sql
  CREATE LOGFILE GROUP mygroup
    ADD UNDOFILE 'new_undo.dat'
    INITIAL_SIZE 2G
  ENGINE NDB;
  ```

  After running the **CREATE LOGFILE GROUP** statement successfully, you should see a result similar to the one shown here for this query against the **FILES** table:

  ```sql
  mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
                FROM INFORMATION_SCHEMA.FILES
               WHERE FILE_NAME = 'new_undo.dat';
  +--------------------+-----------+-----------------------------------------+
  | LOGFILE_GROUP_NAME | FILE_TYPE | EXTRA                                   |
  +--------------------+-----------+-----------------------------------------+
  | mygroup            | UNDO LOG  | CLUSTER_NODE=5;UNDO_BUFFER_SIZE=8388608 |
  | mygroup            | UNDO LOG  | CLUSTER_NODE=6;UNDO_BUFFER_SIZE=8388608 |
  | mygroup            | UNDO LOG  | CLUSTER_NODE=7;UNDO_BUFFER_SIZE=8388608 |
  | mygroup            | UNDO LOG  | CLUSTER_NODE=8;UNDO_BUFFER_SIZE=8388608 |
  +--------------------+-----------+-----------------------------------------+
  ```

  This information was not included in NDB Cluster 8.0 prior to NDB 8.0.15. (Bug #92796, Bug #28800252)

**Notes**

- The **FILES** table is a nonstandard **INFORMATION_SCHEMA** table.

**InnoDB Notes**

The following notes apply to **InnoDB** data files.

- Data reported by **FILES** is reported from the **InnoDB** in-memory cache for open files. By comparison, **INNODB_DATAFILES** reports data from the **InnoDB SYS_DATAFILES** internal data dictionary table.

- The data reported by **FILES** includes global temporary tablespace data. This data is not available in the **InnoDB SYS_DATAFILES** internal data dictionary table, and is therefore not reported by **INNODB_DATAFILES**.
• Undo tablespace data is reported by `FILES` when separate undo tablespaces are present, which they are by default in MySQL 8.0

• The following query returns all data pertinent to InnoDB tablespaces.

```sql
SELECT
  FILE_ID, FILE_NAME, FILE_TYPE, TABLESPACE_NAME, FREE_EXTENTS,
  TOTAL_EXTENTS, EXTENT_SIZE, INITIAL_SIZE, MAXIMUM_SIZE,
  AUTOEXTEND_SIZE, DATA_FREE, STATUS
FROM INFORMATION_SCHEMA.FILES WHERE ENGINE='InnoDB'
```

NDB Notes

• The `FILES` table provides information about Disk Data files only; you cannot use it for determining disk space allocation or availability for individual NDB tables. However, it is possible to see how much space is allocated for each NDB table having data stored on disk—as well as how much remains available for storage of data on disk for that table—using `ndb_desc`. For more information, see `ndb_desc — Describe NDB Tables`.

• The `CREATION_TIME`, `LAST_UPDATE_TIME`, and `LAST_ACCESSED` values are as reported by the operating system, and are not supplied by the NDB storage engine. Where no value is provided by the operating system, these columns display 0000-00-00 00:00:00.

• The difference between the `TOTAL EXTENTS` and `FREE_EXTENTS` columns is the number of extents currently in use by the file:

```sql
SELECT TOTAL_EXTENTS - FREE_EXTENTS AS extents_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

To approximate the amount of disk space in use by the file, multiply that difference by the value of the `EXTENT_SIZE` column, which gives the size of an extent for the file in bytes:

```sql
SELECT (TOTAL_EXTENTS - FREE_EXTENTS) * EXTENT_SIZE AS bytes_used
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

Similarly, you can estimate the amount of space that remains available in a given file by multiplying `FREE_EXTENTS` by `EXTENT_SIZE`:

```sql
SELECT FREE_EXTENTS * EXTENT_SIZE AS bytes_free
FROM INFORMATION_SCHEMA.FILES
WHERE FILE_NAME = 'myfile.dat';
```

**Important**

The byte values produced by the preceding queries are approximations only, and their precision is inversely proportional to the value of `EXTENT_SIZE`. That is, the larger `EXTENT_SIZE` becomes, the less accurate the approximations are.

It is also important to remember that once an extent is used, it cannot be freed again without dropping the data file of which it is a part. This means that deletes from a Disk Data table do not release disk space.

The extent size can be set in a `CREATE TABLESPACE` statement. For more information, see `CREATE TABLESPACE Statement`.

• An additional row is present in the `FILES` table following the creation of a logfile group. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always
that of the `FILE_TYPE` column is always `UNDO FILE`, and that of the `STATUS` column is always `NORMAL`. The value of the `ENGINE` column is always `NDBCLUSTER`.

The `FREE_EXTENTS` column in this row shows the total number of free extents available to all undo files belonging to a given log file group whose name and number are shown in the `LOGFILE_GROUP_NAME` and `LOGFILE_GROUP_NUMBER` columns, respectively.

Suppose there are no existing log file groups on your NDB Cluster, and you create one using the following statement:

```sql
CREATE LOGFILE GROUP lg1
ADD UNDOFILE 'undofile.dat'
INITIAL_SIZE = 16M
UNDO_BUFFER_SIZE = 1M
ENGINE = NDB;
```

You can now see this `NULL` row when you query the `FILES` table:

```sql
SELECT DISTINCT
  FILE_NAME AS File,
  FREE_EXTENTS AS Free,
  TOTAL_EXTENTS AS Total,
  EXTENT_SIZE AS Size,
  INITIAL_SIZE AS Initial
FROM INFORMATION_SCHEMA.FILES;
```

<table>
<thead>
<tr>
<th>File</th>
<th>Free</th>
<th>Total</th>
<th>Size</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>undofile.dat</td>
<td>NULL</td>
<td>4194304</td>
<td>4</td>
<td>16777216</td>
</tr>
<tr>
<td>NULL</td>
<td>4184068</td>
<td>NULL</td>
<td>4</td>
<td>NULL</td>
</tr>
</tbody>
</table>

The total number of free extents available for undo logging is always somewhat less than the sum of the `TOTAL_EXTENTS` column values for all undo files in the log file group due to overhead required for maintaining the undo files. This can be seen by adding a second undo file to the log file group, then repeating the previous query against the `FILES` table:

```sql
ALTER LOGFILE GROUP lg1
ADD UNDOFILE 'undofile02.dat'
INITIAL_SIZE = 4M
ENGINE = NDB;
```

```sql
SELECT DISTINCT
  FILE_NAME AS File,
  FREE_EXTENTS AS Free,
  TOTAL_EXTENTS AS Total,
  EXTENT_SIZE AS Size,
  INITIAL_SIZE AS Initial
FROM INFORMATION_SCHEMA.FILES;
```

<table>
<thead>
<tr>
<th>File</th>
<th>Free</th>
<th>Total</th>
<th>Size</th>
<th>Initial</th>
</tr>
</thead>
<tbody>
<tr>
<td>undofile.dat</td>
<td>NULL</td>
<td>4194304</td>
<td>4</td>
<td>16777216</td>
</tr>
<tr>
<td>undofile02.dat</td>
<td>NULL</td>
<td>1048576</td>
<td>4</td>
<td>4194304</td>
</tr>
<tr>
<td>NULL</td>
<td>5223944</td>
<td>NULL</td>
<td>4</td>
<td>NULL</td>
</tr>
</tbody>
</table>

The amount of free space in bytes which is available for undo logging by Disk Data tables using this log file group can be approximated by multiplying the number of free extents by the initial size:

```sql
SELECT
  FREE_EXTENTS AS 'Free Extents',
  FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
FROM INFORMATION_SCHEMA.FILES;
```
If you create an NDB Cluster Disk Data table and then insert some rows into it, you can see approximately how much space remains for undo logging afterward, for example:

```sql
mysql> CREATE TABLESPACE ts1
    ADD DATAFILE 'data1.dat'
    USE LOGFILE GROUP lg1
    INITIAL_SIZE 512M
    ENGINE = NDB;

mysql> CREATE TABLE dd (c1 INT NOT NULL PRIMARY KEY,
    c2 INT,
    c3 DATE)
    TABLESPACE ts1 STORAGE DISK
    ENGINE = NDB;

mysql> INSERT INTO dd VALUES
    (NULL, 1234567890, '2007-02-02'),
    (NULL, 1126789005, '2007-02-03'),
    (NULL, 1357924680, '2007-02-04'),
    (NULL, 1642097531, '2007-02-05');

mysql> SELECT FREE_EXTENTS AS 'Free Extents',
    FREE_EXTENTS * EXTENT_SIZE AS 'Free Bytes'
    FROM INFORMATION_SCHEMA.FILES
    WHERE LOGFILE_GROUP_NAME = 'lg1'
    AND FILE_NAME IS NULL;
```

+--------------+------------+
| Free Extents | Free Bytes |
+--------------+------------+
| 5207565      | 20830260   |
+--------------+------------+

- An additional row is present in the `FILES` table for any NDB Cluster tablespace, whether or not any data files are associated with the tablespace. This row has `NULL` for the value of the `FILE_NAME` column. For this row, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `TABLESPACE`, and that of the `STATUS` column is always `NORMAL`. The value of the `ENGINE` column is always `NDBCLUSTER`.

- For additional information, and examples of creating and dropping NDB Cluster Disk Data objects, see [NDB Cluster Disk Data Tables](#).
Chapter 23 The INFORMATION_SCHEMA PROCESSLIST Table

The `PROCESSLIST` table provides information about which threads are running.

The `PROCESSLIST` table has these columns:

- **ID**
  The connection identifier. This is the same type of value displayed in the `Id` column of the `SHOW PROCESSLIST` statement, the `PROCESSLIST_ID` column of the Performance Schema `threads` table, and returned by the `CONNECTION_ID()` function.

- **USER**
  The MySQL user who issued the statement. A value of `system user` refers to a nonclient thread spawned by the server to handle tasks internally. This could be the I/O or SQL thread used on replication slaves or a delayed-row handler. For `system user`, there is no host specified in the `Host` column. `unauthenticated user` refers to a thread that has become associated with a client connection but for which authentication of the client user has not yet been done. `event_scheduler` refers to the thread that monitors scheduled events (see Using the Event Scheduler).

  **Note**
  A `USER` value of `system user` is distinct from the `SYSTEM_USER` privilege. The former designates internal threads. The latter distinguishes the system user and regular user account categories (see Account Categories).

- **HOST**
  The host name of the client issuing the statement (except for `system user`, for which there is no host). The host name for TCP/IP connections is reported in `host_name:client_port` format to make it easier to determine which client is doing what.

- **DB**
  The default database, if one is selected; otherwise `NULL`.

- **COMMAND**
  The type of command the thread is executing. For descriptions for thread commands, see Examining Thread Information. The value of this column corresponds to the `COM_`xxx` commands of the client/server protocol and `Com_`xxx` status variables. See Server Status Variables.

- **TIME**
  The time in seconds that the thread has been in its current state. For a slave SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the slave machine. See Replication Implementation Details.

- **STATE**
  An action, event, or state that indicates what the thread is doing. Descriptions for `STATE` values can be found at Examining Thread Information.

  Most states correspond to very quick operations. If a thread stays in a given state for many seconds, there might be a problem that needs to be investigated.
For the `SHOW PROCESSLIST` statement, the value of `STATE` is `NULL`.

- **INFO**

  The statement the thread is executing, or `NULL` if it is not executing any statement. The statement might be the one sent to the server, or an innermost statement if the statement executes other statements. For example, if a `CALL` statement executes a stored procedure that is executing a `SELECT` statement, the `INFO` value shows the `SELECT` statement.

**Notes**

- The `PROCESSLIST` table is a nonstandard `INFORMATION_SCHEMA` table.

- Like the output from the `SHOW PROCESSLIST` statement, the `PROCESSLIST` table shows information only about your own threads, unless you have the `PROCESS` privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.

- If an SQL statement refers to the `PROCESSLIST` table, MySQL populates the entire table once, when statement execution begins, so there is read consistency during the statement. There is no read consistency for a multi-statement transaction.

Process information is also available from the `mysqladmin processlist` command, the `SHOW PROCESSLIST` statement, and the Performance Schema `threads` table (see `mysqladmin` — Client for Administering a MySQL Server, `SHOW PROCESSLIST Statement`, and `The threads Table`). In contrast to the `INFORMATION_SCHEMA.PROCESSLIST` table and `SHOW PROCESSLIST` statement, which have negative performance consequences because they require a mutex, access to `threads` does not require a mutex and has minimal impact on server performance. The `threads` table also shows information about background threads, which the `PROCESSLIST` table and `SHOW PROCESSLIST` do not. This means that `threads` can be used to monitor activity the other thread information sources cannot.

The following statements are equivalent:

```sql
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```
Chapter 24 The INFORMATION_SCHEMA
REFERENTIAL_CONSTRAINTS Table

The REFERENTIAL_CONSTRAINTS table provides information about foreign keys.

The REFERENTIAL_CONSTRAINTS table has these columns:

- **CONSTRAINT_CATALOG**
  The name of the catalog to which the constraint belongs. This value is always `def`.

- **CONSTRAINT_SCHEMA**
  The name of the schema (database) to which the constraint belongs.

- **CONSTRAINT_NAME**
  The name of the constraint.

- **UNIQUE_CONSTRAINT_CATALOG**
  The name of the catalog containing the unique constraint that the constraint references. This value is always `def`.

- **UNIQUE_CONSTRAINT_SCHEMA**
  The name of the schema containing the unique constraint that the constraint references.

- **UNIQUE_CONSTRAINT_NAME**
  The name of the unique constraint that the constraint references.

- **MATCH_OPTION**
  The value of the constraint `MATCH` attribute. The only valid value at this time is `NONE`.

- **UPDATE_RULE**
  The value of the constraint `ON UPDATE` attribute. The possible values are `CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION`.

- **DELETE_RULE**
  The value of the constraint `ON DELETE` attribute. The possible values are `CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION`.

- **TABLE_NAME**
  The name of the table. This value is the same as in the TABLE_CONSTRAINTS table.

- **REFERENCED_TABLE_NAME**
  The name of the table referenced by the constraint.
Some extensions to `SHOW` statements accompany the implementation of `INFORMATION_SCHEMA`:

- `SHOW` can be used to get information about the structure of `INFORMATION_SCHEMA` itself.
- Several `SHOW` statements accept a `WHERE` clause that provides more flexibility in specifying which rows to display.

`INFORMATION_SCHEMA` is an information database, so its name is included in the output from `SHOW DATABASES`. Similarly, `SHOW TABLES` can be used with `INFORMATION_SCHEMA` to obtain a list of its tables:

```sql
mysql> SHOW TABLES FROM INFORMATION_SCHEMA;
+---------------------------------------+
| Tables_in_INFORMATION_SCHEMA          |
+---------------------------------------+
| CHARACTER_SETS                        |
| COLLATIONS                            |
| COLLATION_CHARACTER_SET_APPLICABILITY |
| COLUMNS                               |
| COLUMN_PRIVILEGES                     |
| ENGINES                               |
| EVENTS                                |
| FILES                                 |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| STATISTICS                             |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
```

`SHOW COLUMNS` and `DESCRIBE` can display information about the columns in individual `INFORMATION_SCHEMA` tables.

`SHOW` statements that accept a `LIKE` clause to limit the rows displayed also permit a `WHERE` clause that specifies more general conditions that selected rows must satisfy:

```sql
SHOW CHARACTER SET
SHOW COLLATION
SHOW COLUMNS
SHOW DATABASES
SHOW FUNCTION STATUS
SHOW INDEX
SHOW OPEN TABLES
SHOW PROCEDURE STATUS
SHOW STATUS
SHOW TABLE STATUS
SHOW TABLES
SHOW TRIGGERS
SHOW VARIABLES
```

The `WHERE` clause, if present, is evaluated against the column names displayed by the `SHOW` statement. For example, the `SHOW CHARACTER_SET` statement produces these output columns:
To use a `WHERE` clause with `SHOW CHARACTER SET`, you would refer to those column names. As an example, the following statement displays information about character sets for which the default collation contains the string 'japanese':

```
mysql> SHOW CHARACTER SET WHERE Default collation LIKE '%japanese%';
```

This statement displays the multibyte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
```
Chapter 26 MySQL 8.0 FAQ: INFORMATION_SCHEMA

Questions

• 26.1: Where can I find documentation for the MySQL INFORMATION_SCHEMA database?

See Chapter 1, INFORMATION_SCHEMA Tables

• 26.2: Is there a discussion forum for INFORMATION_SCHEMA?


• 26.3: Where can I find the ANSI SQL 2003 specification for INFORMATION_SCHEMA?

Unfortunately, the official specifications are not freely available. (ANSI makes them available for purchase.) However, there are books available, such as SQL-99 Complete, Really by Peter Gulutzan and Trudy Pelzer, that provide a comprehensive overview of the standard, including INFORMATION_SCHEMA.

• 26.4: What is the difference between the Oracle Data Dictionary and MySQL INFORMATION_SCHEMA?

Both Oracle and MySQL provide metadata in tables. However, Oracle and MySQL use different table names and column names. The MySQL implementation is more similar to those found in DB2 and SQL Server, which also support INFORMATION_SCHEMA as defined in the SQL standard.

• 26.5: Can I add to or otherwise modify the tables found in the INFORMATION_SCHEMA database?

No. Since applications may rely on a certain standard structure, this should not be modified. For this reason, we cannot support bugs or other issues which result from modifying INFORMATION_SCHEMA tables or data.