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: 2021-10-22 (revision: 71169)

Table of Contents

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

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 for Accessibility

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 Introduction

`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*.

- INFORMATION_SCHEMA Usage Notes
- Character Set Considerations
- INFORMATION_SCHEMA as Alternative to SHOW Statements
- INFORMATION_SCHEMA and Privileges
- Performance Considerations
- Standards Considerations
- Conventions in the INFORMATION_SCHEMA Reference Sections
- Related Information

INFORMATION_SCHEMA Usage Notes

`INFORMATION_SCHEMA` is a database within each MySQL instance, the place that stores information about all the other databases that the MySQL server maintains. The `INFORMATION_SCHEMA` database contains several read-only tables. They are actually views, not base tables, so there are no files associated with them, and you cannot set triggers on them. Also, there is no database directory with that name.

Although you can select `INFORMATION_SCHEMA` as the default database with a `USE` statement, you can only read the contents of tables, not perform `INSERT`, `UPDATE`, or `DELETE` operations on them.

Here is an example of a statement that retrieves information from `INFORMATION_SCHEMA`:

```
mysql> SELECT table_name, table_type, engine
       FROM information_schema.tables
       WHERE table_schema = 'db5'
       ORDER BY table_name;
+------------+------------+--------+
| table_name | table_type | engine |
+------------+------------+--------+
| fk         | BASE TABLE | InnoDB |
| fk2        | BASE TABLE | InnoDB |
| goto       | BASE TABLE | MyISAM |
| into       | BASE TABLE | MyISAM |
| k          | BASE TABLE | MyISAM |
| kurs       | BASE TABLE | MyISAM |
| loop       | BASE TABLE | MyISAM |
| pk         | BASE TABLE | InnoDB |
| t          | BASE TABLE | MyISAM |
| t2         | BASE TABLE | MyISAM |
| t3         | BASE TABLE | MyISAM |
| t7         | BASE TABLE | MyISAM |
| tables     | BASE TABLE | MyISAM |
| v          | VIEW       | NULL   |
| v2         | VIEW       | NULL   |
| v3         | VIEW       | NULL   |
| v56        | VIEW       | NULL   |
+------------+------------+--------+
17 rows in set (0.01 sec)
```

Explanation: The statement requests a list of all the tables in database `db5`, showing just three pieces of information: the name of the table, its type, and its storage engine.

Character Set Considerations

The definition for character columns (for example, `TABLES.TABLE_NAME`) is generally `VARCHAR(N) CHARACTER SET utf8` where *`N`* is at least 64. MySQL uses the default collation for this character set (`utf8_general_ci`) for all searches, sorts, comparisons, and other string operations on such columns.

Because some MySQL objects are represented as files, searches in `INFORMATION_SCHEMA` string columns can be affected by file system case sensitivity. For more information, see Using Collation in INFORMATION_SCHEMA Searches.

INFORMATION_SCHEMA as Alternative to SHOW Statements

The `SELECT ... FROM INFORMATION_SCHEMA` statement is intended as a more consistent way to provide access to the information provided by the various `SHOW` statements that MySQL supports (`SHOW DATABASES`, `SHOW TABLES`, and so forth). Using `SELECT` has these advantages, compared to `SHOW`:

- It conforms to Codd's rules, because all access is done on tables.
- You can use the familiar syntax of the `SELECT` statement, and only need to learn some table and column names.
- The implementor need not worry about adding keywords.
- You can filter, sort, concatenate, and transform the results from `INFORMATION_SCHEMA` queries into whatever format your application needs, such as a data structure or a text representation to parse.
- This technique is more interoperable with other database systems. For example, Oracle Database users are familiar with querying tables in the Oracle data dictionary.

Because `SHOW` is familiar and widely used, the `SHOW` statements remain as an alternative. In fact, along with the implementation of `INFORMATION_SCHEMA`, there are enhancements to `SHOW` as described in Chapter 9, *Extensions to SHOW Statements*.

INFORMATION_SCHEMA and Privileges

For most `INFORMATION_SCHEMA` tables, each MySQL user has the right to access them, but can see only the rows in the tables that correspond to objects for which the user has the proper access privileges. In some cases (for example, the `ROUTINE_DEFINITION` column in the `INFORMATION_SCHEMA ROUTINES` table), users who have insufficient privileges see `NULL`. Some tables have different privilege requirements; for these, the requirements are mentioned in the applicable table descriptions. For example, `InnoDB` tables (tables with names that begin with `INNODB_`) require the `PROCESS` privilege.

The same privileges apply to selecting information from `INFORMATION_SCHEMA` and viewing the same information through `SHOW` statements. In either case, you must have some privilege on an object to see information about it.

Performance Considerations

`INFORMATION_SCHEMA` queries that search for information from more than one database might take a long time and impact performance. To check the efficiency of a query, you can use `EXPLAIN`. For information about using `EXPLAIN` output to tune `INFORMATION_SCHEMA` queries, see Optimizing INFORMATION_SCHEMA Queries.

Standards Considerations

The implementation for the INFORMATION_SCHEMA table structures in MySQL follows the ANSI/ISO SQL:2003 standard Part 11 *Schemata*. Our intent is approximate compliance with SQL:2003 core feature F021 *Basic information schema*.

Users of SQL Server 2000 (which also follows the standard) may notice a strong similarity. However, MySQL has omitted many columns that are not relevant for our implementation, and added columns that are MySQL-specific. One such added column is the ENGINE column in the INFORMATION_SCHEMA TABLES table.

Although other DBMSs use a variety of names, like syscat or system, the standard name is INFORMATION_SCHEMA.

To avoid using any name that is reserved in the standard or in DB2, SQL Server, or Oracle, we changed the names of some columns marked “MySQL extension”. (For example, we changed COLLATION to TABLE_COLLATION in the TABLES table.) See the list of reserved words near the end of this article: https://web.archive.org/web/20070428032454/http://www.dbazine.com/db2/db2-disarticles/gulutzan5.

Conventions in the INFORMATION_SCHEMA Reference Sections

The following sections describe each of the tables and columns in INFORMATION_SCHEMA. For each column, there are three pieces of information:

- “INFORMATION_SCHEMA Name” indicates the name for the column in the INFORMATION_SCHEMA table. This corresponds to the standard SQL name unless the “Remarks” field says “MySQL extension.”
- “SHOW Name” indicates the equivalent field name in the closest SHOW statement, if there is one.
- “Remarks” provides additional information where applicable. If this field is NULL, it means that the value of the column is always NULL. If this field says “MySQL extension,” the column is a MySQL extension to standard SQL.

Many sections indicate what SHOW statement is equivalent to a SELECT that retrieves information from INFORMATION_SCHEMA. For SHOW statements that display information for the default database if you omit a FROM *db_name* clause, you can often select information for the default database by adding an AND TABLE_SCHEMA = SCHEMA() condition to the WHERE clause of a query that retrieves information from an INFORMATION_SCHEMA table.

Related Information

These sections discuss additional INFORMATION_SCHEMA-related topics:

- information about INFORMATION_SCHEMA tables specific to the InnoDB storage engine: Chapter 5, *INFORMATION_SCHEMA InnoDB Tables*
- information about INFORMATION_SCHEMA tables specific to the thread pool plugin: Chapter 6, *INFORMATION_SCHEMA Thread Pool Tables*
- information about INFORMATION_SCHEMA tables specific to the CONNECTION_CONTROL plugin: Chapter 7, *INFORMATION_SCHEMA Connection-Control Tables*
- Answers to questions that are often asked concerning the INFORMATION_SCHEMA database: Chapter 10, *MySQL 8.0 FAQ: INFORMATION_SCHEMA*
- INFORMATION_SCHEMA queries and the optimizer: Optimizing INFORMATION_SCHEMA Queries

- The effect of collation on `INFORMATION_SCHEMA` comparisons: Using Collation in INFORMATION_SCHEMA Searches

Chapter 3 INFORMATION_SCHEMA Table Reference

The following table summarizes all available `INFORMATION_SCHEMA` tables. For greater detail, see the individual table descriptions.

Table 3.1 INFORMATION_SCHEMA Tables

Table Name	Description	Introduced	Deprecated
`ADMINISTRABLE_ROLE_AUTHORIZATIONS`	Grantable users or roles for current user or role	8.0.19	
`APPLICABLE_ROLES`	Applicable roles for current user	8.0.19	
`CHARACTER_SETS`	Available character sets		
`CHECK_CONSTRAINTS`	Table and column CHECK constraints	8.0.16	
`COLLATION_CHARACTER_SET_APPLICABILITY`	Character set applicable to each collation		
`COLLATIONS`	Collations for each character set		
`COLUMN_PRIVILEGES`	Privileges defined on columns		
`COLUMN_STATISTICS`	Histogram statistics for column values		
`COLUMNS`	Columns in each table		
`COLUMNS_EXTENSIONS`	Column attributes for primary and secondary storage engines	8.0.21	
`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`	Current number of consecutive failed connection attempts per account		
`ENABLED_ROLES`	Roles enabled within current session	8.0.19	
`ENGINES`	Storage engine properties		
`EVENTS`	Event Manager events		
`FILES`	Files that store tablespace data		
`INNODB_BUFFER_PAGE`	Pages in InnoDB buffer pool		
`INNODB_BUFFER_PAGE_LRU`	LRU ordering of pages in InnoDB buffer pool		
`INNODB_BUFFER_POOL_STATS`	InnoDB buffer pool statistics		
`INNODB_CACHED_INDEXES`	Number of index pages cached per index in InnoDB buffer pool		

Table Name	Description	Introduced	Deprecated
INNODB_CMP	Status for operations related to compressed InnoDB tables		
INNODB_CMP_PER_INDEX	Status for operations related to compressed InnoDB tables and indexes		
INNODB_CMP_PER_INDEX_RESET	Status for operations related to compressed InnoDB tables and indexes		
INNODB_CMP_RESET	Status for operations related to compressed InnoDB tables		
INNODB_CMPMEM	Status for compressed pages within InnoDB buffer pool		
INNODB_CMPMEM_RESET	Status for compressed pages within InnoDB buffer pool		
INNODB_COLUMNS	Columns in each InnoDB table		
INNODB_DATAFILES	Data file path information for InnoDB file-per-table and general tablespaces		
INNODB_FIELDS	Key columns of InnoDB indexes		
INNODB_FOREIGN	InnoDB foreign-key metadata		
INNODB_FOREIGN_COLS	InnoDB foreign-key column status information		
INNODB_FT_BEING_DELETED	Snapshot of INNODB_FT_DELETED table		
INNODB_FT_CONFIG	Metadata for InnoDB table FULLTEXT index and associated processing		
INNODB_FT_DEFAULT_STOPWORD	Default list of stopwords for InnoDB FULLTEXT indexes		
INNODB_FT_DELETED	Rows deleted from InnoDB table FULLTEXT index		

Table Name	Description	Introduced	Deprecated
INNODB_FT_INDEX_CACHE	Token information for newly inserted rows in InnoDB FULLTEXT index		
INNODB_FT_INDEX_TABLE	Inverted index information for processing text searches against InnoDB table FULLTEXT index		
INNODB_INDEXES	InnoDB index metadata		
INNODB_METRICS	InnoDB performance information		
INNODB_SESSION_TEMP_TABLESPACES	Session temporary-tablespace metadata	8.0.13	
INNODB_TABLES	InnoDB table metadata		
INNODB_TABLESPACES	InnoDB file-per-table, general, and undo tablespace metadata		
INNODB_TABLESPACES_BRIEF	Brief file-per-table, general, undo, and system tablespace metadata		
INNODB_TABLESTATS	InnoDB table low-level status information		
INNODB_TEMP_TABLE_INFO	Information about active user-created InnoDB temporary tables		
INNODB_TRX	Active InnoDB transaction information		
INNODB_VIRTUAL	InnoDB virtual generated column metadata		
KEY_COLUMN_USAGE	Which key columns have constraints		
KEYWORDS	MySQL keywords		
MYSQL_FIREWALL_USERS	Firewall in-memory data for account profiles		8.0.26
MYSQL_FIREWALL_WHITELIST	Firewall in-memory data for account profile allowlists		8.0.26
ndb_transid_mysql_connection_map	NDB transaction information		
OPTIMIZER_TRACE	Information produced by optimizer trace activity		
PARAMETERS	Stored routine parameters and stored function return values		

Table Name	Description	Introduced	Deprecated
PARTITIONS	Table partition information		
PLUGINS	Plugin information		
PROCESSLIST	Information about currently executing threads		
PROFILING	Statement profiling information		
REFERENTIAL_CONSTRAINTS	Foreign key information		
RESOURCE_GROUPS	Resource group information		
ROLE_COLUMN_GRANTS	Column privileges for roles available to or granted by currently enabled roles	8.0.19	
ROLE_ROUTINE_GRANTS	Routine privileges for roles available to or granted by currently enabled roles	8.0.19	
ROLE_TABLE_GRANTS	Table privileges for roles available to or granted by currently enabled roles	8.0.19	
ROUTINES	Stored routine information		
SCHEMA_PRIVILEGES	Privileges defined on schemas		
SCHEMATA	Schema information		
SCHEMATA_EXTENSIONS	Schema options	8.0.22	
ST_GEOMETRY_COLUMNS	Columns in each table that store spatial data		
ST_SPATIAL_REFERENCE_SYSTEMS	Available spatial reference systems		
ST_UNITS_OF_MEASURE	Acceptable units for ST_Distance()	8.0.14	
STATISTICS	Table index statistics		
TABLE_CONSTRAINTS	Which tables have constraints		
TABLE_CONSTRAINTS_EXTENSIONS	Table constraint attributes for primary and secondary storage engines	8.0.21	
TABLE_PRIVILEGES	Privileges defined on tables		
TABLES	Table information		

Table Name	Description	Introduced	Deprecated
TABLES_EXTENSIONS	Table attributes for primary and secondary storage engines	8.0.21	
TABLESPACES	Tablespace information		
TABLESPACES_EXTENSIONS	Tablespace attributes for primary storage engines	8.0.21	
TP_THREAD_GROUP_STATE	Thread pool thread group states		
TP_THREAD_GROUP_STATS	Thread pool thread group statistics		
TP_THREAD_STATE	Thread pool thread information		
TRIGGERS	Trigger information		
USER_ATTRIBUTES	User comments and attributes	8.0.21	
USER_PRIVILEGES	Privileges defined globally per user		
VIEW_ROUTINE_USAGE	Stored functions used in views	8.0.13	
VIEW_TABLE_USAGE	Tables and views used in views	8.0.13	
VIEWS	View information		

Chapter 4 INFORMATION_SCHEMA General Tables

Table of Contents

The following sections describe what may be denoted as the “general” set of `INFORMATION_SCHEMA` tables. These are the tables not associated with particular storage engines, components, or plugins.

4.1 INFORMATION_SCHEMA General Table Reference

The following table summarizes `INFORMATION_SCHEMA` general tables. For greater detail, see the individual table descriptions.

Table 4.1 INFORMATION_SCHEMA General Tables

Table Name	Description	Introduced
`ADMINISTRABLE_ROLE_AUTHORIZATIONS`	Grantable users or roles for current user or role	8.0.19
`APPLICABLE_ROLES`	Applicable roles for current user	8.0.19
`CHARACTER_SETS`	Available character sets	
`CHECK_CONSTRAINTS`	Table and column CHECK constraints	8.0.16
`COLLATION_CHARACTER_SET_APPLICABILITY`	Character set applicable to each collation	
`COLLATIONS`	Collations for each character set	
`COLUMN_PRIVILEGES`	Privileges defined on columns	
`COLUMN_STATISTICS`	Histogram statistics for column values	
`COLUMNS`	Columns in each table	
`COLUMNS_EXTENSIONS`	Column attributes for primary and secondary storage engines	8.0.21
`ENABLED_ROLES`	Roles enabled within current session	8.0.19
`ENGINES`	Storage engine properties	
`EVENTS`	Event Manager events	
`FILES`	Files that store tablespace data	
`KEY_COLUMN_USAGE`	Which key columns have constraints	
`KEYWORDS`	MySQL keywords	
`ndb_transid_mysql_connection_map`	NDB transaction information	
`OPTIMIZER_TRACE`	Information produced by optimizer trace activity	
`PARAMETERS`	Stored routine parameters and stored function return values	
`PARTITIONS`	Table partition information	
`PLUGINS`	Plugin information	
`PROCESSLIST`	Information about currently executing threads	

Table Name	Description	Introduced
PROFILING	Statement profiling information	
REFERENTIAL_CONSTRAINTS	Foreign key information	
RESOURCE_GROUPS	Resource group information	
ROLE_COLUMN_GRANTS	Column privileges for roles available to or granted by currently enabled roles	8.0.19
ROLE_ROUTINE_GRANTS	Routine privileges for roles available to or granted by currently enabled roles	8.0.19
ROLE_TABLE_GRANTS	Table privileges for roles available to or granted by currently enabled roles	8.0.19
ROUTINES	Stored routine information	
SCHEMA_PRIVILEGES	Privileges defined on schemas	
SCHEMATA	Schema information	
SCHEMATA_EXTENSIONS	Schema options	8.0.22
ST_GEOMETRY_COLUMNS	Columns in each table that store spatial data	
ST_SPATIAL_REFERENCE_SYSTEMS	Available spatial reference systems	
ST_UNITS_OF_MEASURE	Acceptable units for ST_Distance()	8.0.14
STATISTICS	Table index statistics	
TABLE_CONSTRAINTS	Which tables have constraints	
TABLE_CONSTRAINTS_EXTENSIONS	Table constraint attributes for primary and secondary storage engines	8.0.21
TABLE_PRIVILEGES	Privileges defined on tables	
TABLES	Table information	
TABLES_EXTENSIONS	Table attributes for primary and secondary storage engines	8.0.21
TABLESPACES	Tablespace information	
TABLESPACES_EXTENSIONS	Tablespace attributes for primary storage engines	8.0.21
TRIGGERS	Trigger information	
USER_ATTRIBUTES	User comments and attributes	8.0.21
USER_PRIVILEGES	Privileges defined globally per user	
VIEW_ROUTINE_USAGE	Stored functions used in views	8.0.13
VIEW_TABLE_USAGE	Tables and views used in views	8.0.13
VIEWS	View information	

4.2 The INFORMATION_SCHEMA ADMINISTRABLE_ROLE_AUTHORIZATIONS Table

The `ADMINISTRABLE_ROLE_AUTHORIZATIONS` table (available as of MySQL 8.0.19) provides information about which roles applicable for the current user or role can be granted to other users or roles.

The `ADMINISTRABLE_ROLE_AUTHORIZATIONS` table has these columns:

- `USER`

  The user name part of the current user account.

- `HOST`

  The host name part of the current user account.

- `GRANTEE`

  The user name part of the account to which the role is granted.

- `GRANTEE_HOST`

  The host name part of the account to which the role is granted.

- `ROLE_NAME`

  The user name part of the granted role.

- `ROLE_HOST`

  The host name part of the granted role.

- `IS_GRANTABLE`

  `YES` or `NO`, depending on whether the role is grantable to other accounts.

- `IS_DEFAULT`

  `YES` or `NO`, depending on whether the role is a default role.

- `IS_MANDATORY`

  `YES` or `NO`, depending on whether the role is mandatory.

4.3 The INFORMATION_SCHEMA APPLICABLE_ROLES Table

The `APPLICABLE_ROLES` table (available as of MySQL 8.0.19) provides information about the roles that are applicable for the current user.

The `APPLICABLE_ROLES` table has these columns:

- `USER`

  The user name part of the current user account.

- `HOST`

  The host name part of the current user account.

- `GRANTEE`

  The user name part of the account to which the role is granted.

- `GRANTEE_HOST`

  The host name part of the account to which the role is granted.

- `ROLE_NAME`

  The user name part of the granted role.

- `ROLE_HOST`

  The host name part of the granted role.

- `IS_GRANTABLE`

  `YES` or `NO`, depending on whether the role is grantable to other accounts.

- `IS_DEFAULT`

  `YES` or `NO`, depending on whether the role is a default role.

- `IS_MANDATORY`

  `YES` or `NO`, depending on whether the role is mandatory.

4.4 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:

```
SELECT * FROM INFORMATION_SCHEMA.CHARACTER_SETS
  [WHERE CHARACTER_SET_NAME LIKE 'wild']
SHOW CHARACTER SET
  [LIKE 'wild']
```

4.5 The INFORMATION_SCHEMA CHECK_CONSTRAINTS Table

As of MySQL 8.0.16, `CREATE TABLE` permits the core features of table and column `CHECK` constraints, and the the `CHECK_CONSTRAINTS` table provides information about these constraints.

The `CHECK_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.

- `CHECK_CLAUSE`

  The expression that specifies the constraint condition.

4.6 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, either `NO PAD` or `PAD SPACE`. This attribute affects whether trailing spaces are significant in string comparisons; see Trailing Space Handling in Comparisons.

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']
```

4.7 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.

4.8 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 Section 4.35, “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.

- `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:

```
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']
```

4.9 The INFORMATION_SCHEMA COLUMNS_EXTENSIONS Table

The `COLUMNS_EXTENSIONS` table (available as of MySQL 8.0.21) provides information about column attributes defined for primary and secondary storage engines.

> **Note**
>
> The `COLUMNS_EXTENSIONS` table is reserved for future use.

The `COLUMNS_EXTENSIONS` 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.

- `COLUMN_NAME`

  The name of the column.

- `ENGINE_ATTRIBUTE`

  Column attributes defined for the primary storage engine. Reserved for future use.

- `SECONDARY_ENGINE_ATTRIBUTE`

  Column attributes defined for the secondary storage engine. Reserved for future use.

4.10 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

- COLUMN_PRIVILEGES is a nonstandard INFORMATION_SCHEMA table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.COLUMN_PRIVILEGES
SHOW GRANTS ...
```

4.11 The INFORMATION_SCHEMA COLUMN_STATISTICS Table

The COLUMN_STATISTICS table provides access to histogram statistics for column values.

For information about histogram statistics, see Optimizer Statistics, and ANALYZE TABLE Statement.

You can see information only for columns for which you have some privilege.

The COLUMN_STATISTICS table has these columns:

- SCHEMA_NAME

  The names of the schema for which the statistics apply.

- TABLE_NAME

The names of the column for which the statistics apply.

- `COLUMN_NAME`

  The names of the column for which the statistics apply.

- `HISTOGRAM`

  A `JSON` object describing the column statistics, stored as a histogram.

4.12 The INFORMATION_SCHEMA ENABLED_ROLES Table

The `ENABLED_ROLES` table (available as of MySQL 8.0.19) provides information about the roles that are enabled within the current session.

The `ENABLED_ROLES` table has these columns:

- `ROLE_NAME`

  The user name part of the granted role.

- `ROLE_HOST`

  The host name part of the granted role.

- `IS_DEFAULT`

  `YES` or `NO`, depending on whether the role is a default role.

- `IS_MANDATORY`

  `YES` or `NO`, depending on whether the role is mandatory.

4.13 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.

Value	Meaning
`YES`	The engine is supported and is active
`DEFAULT`	Like `YES`, plus this is the default engine
`NO`	The engine is not supported
`DISABLED`	The engine is supported but has been disabled

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

- ENGINES 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:

```
SELECT * FROM INFORMATION_SCHEMA.ENGINES
SHOW ENGINES
```

4.14 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 source and replicated to the current MySQL server which is acting as a replica, but the event is not presently being executed on the replica. For more information, see Replication of Invoked Features. information.

- 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 replication source. 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

- EVENTS 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:

```
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:

```
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']
```

4.15 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`. In NDB 8.0, the file name is shown with a relative path; for an undo log file, this path is relative to the directory `DataDir/ndb_NodeId_fs/LG`; for a data file, it is relative to the directory `DataDir/ndb_NodeId_fs/TS`. This means, for example, that the name of a data file created with `ALTER TABLESPACE ts ADD DATAFILE 'data_2.dat' INITIAL SIZE 256M` is shown as `./data_2.dat`.

- `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` or `DATAFILE`. Prior to NDB 8.0.13, `TABLESPACE` was also a possible value.

- `TABLESPACE_NAME`

  The name of the tablespace with which the file is associated.

  For `InnoDB`: General tablespace names are as specified when created. File-per-table tablespace names are shown in the following format: *`schema_name/table_name`*. The `InnoDB` system tablespace name

is `innodb_system`. The global temporary tablespace name is `innodb_temporary`. Default undo tablespace names are `innodb_undo_001` and `innodb_undo_002`. User-created undo tablespace names are as specified when created.

- `TABLE_CATALOG`

  This value is always empty.

- `TABLE_SCHEMA`

  This is always `NULL`.

- `TABLE_NAME`

  This is always `NULL`.

- `LOGFILE_GROUP_NAME`

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

  For `NDB`: For a Disk Data undo log file, the auto-generated ID number of the log file group to which the log file belongs. This is the same as the value shown for the `id` column in the `ndbinfo.dict_obj_info` table and the `log_id` column in the `ndbinfo.logspaces` and `ndbinfo.logspaces` tables for this undo log file.

- `ENGINE`

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

  For `NDB`: This value is always `ndbcluster`.

- `FULLTEXT_KEYS`

  This is always `NULL`.

- `DELETED_ROWS`

  This is always `NULL`.

- `UPDATE_COUNT`

  This 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: This value is always the same as the INITIAL_SIZE value.

- AUTOEXTEND_SIZE

  The auto-extend size of the tablespace. For NDB, AUTOEXTEND_SIZE is always NULL.

- CREATION_TIME

  This is always NULL.

- LAST_UPDATE_TIME

  This is always NULL.

- LAST_ACCESS_TIME

  This is always NULL.

- RECOVER_TIME

  This is always NULL.

- TRANSACTION_COUNTER

  This is always NULL.

- VERSION

  For InnoDB: This is always NULL.

For NDB: The version number of the file.

- ROW_FORMAT

  For InnoDB: This is always NULL.

  For NDB: One of FIXED or DYNAMIC.

- TABLE_ROWS

  This is always NULL.

- AVG_ROW_LENGTH

  This is always NULL.

- DATA_LENGTH

  This is always NULL.

- MAX_DATA_LENGTH

  This is always NULL.

- INDEX_LENGTH

  This 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: This is always NULL.

- CREATE_TIME

  This is always NULL.

- UPDATE_TIME

  This is always NULL.

- CHECK_TIME

  This is always NULL.

- CHECKSUM

  This 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 is always NULL.

For NDB: (*NDB 8.0.15 and later*) For undo log files, this column shows the undo log buffer size; for data files, it is always *NULL*. A more detailed explanation is provided in the next few paragraphs.

NDBCLUSTER stores a copy of each data file and each undo log file on each data node in the cluster. In NDB 8.0.13 and later, the FILES table contains only one row for each such file. Suppose that you run the following two statements on an NDB Cluster with four data nodes:

```
CREATE LOGFILE GROUP mygroup
    ADD UNDOFILE 'new_undo.dat'
    INITIAL_SIZE 2G
    ENGINE NDBCLUSTER;
CREATE TABLESPACE myts
    ADD DATAFILE 'data_1.dat'
    USE LOGFILE GROUP mygroup
    INITIAL_SIZE 256M
    ENGINE NDBCLUSTER;
```

After running these two statements successfully, you should see a result similar to the one shown here for this query against the FILES table:

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
    ->     FROM INFORMATION_SCHEMA.FILES
    ->     WHERE ENGINE = 'ndbcluster';
+--------------------+-----------+-------------------------+
| LOGFILE_GROUP_NAME | FILE_TYPE | EXTRA                   |
+--------------------+-----------+-------------------------+
| mygroup            | UNDO LOG  | UNDO_BUFFER_SIZE=8388608 |
| mygroup            | DATAFILE  | NULL                    |
+--------------------+-----------+-------------------------+
```

The undo log buffer size information was inadvertently removed in NDB 8.0.13, but was restored in NDB 8.0.15. (Bug #92796, Bug #28800252)

Prior to NDB 8.0.13, the FILES table contained a row for each of these files on each data node the file belonged to, as well as the size of its undo buffer. In these versions, the result of the same query contains one row per data node, as shown here:

```
+--------------------+-----------+----------------------------------------+
| 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 |
| mygroup            | DATAFILE  | CLUSTER_NODE=5                         |
| mygroup            | DATAFILE  | CLUSTER_NODE=6                         |
| mygroup            | DATAFILE  | CLUSTER_NODE=7                         |
| mygroup            | DATAFILE  | CLUSTER_NODE=8                         |
+--------------------+-----------+----------------------------------------+
```

Notes

- FILES is a nonstandard INFORMATION_SCHEMA table.
- As of MySQL 8.0.21, you must have the PROCESS privilege to query this 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.

```
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'\G
```

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

```
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:

```
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`:

```
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.

- Prior to NDB 8.0.13, an additional row was present in the `FILES` table following the creation of a logfile group, having `NULL` in the `FILE_NAME` column. In NDB 8.0.13 and later, this row— which did not correspond to any file—is no longer shown, and it is necessary to query the `ndbinfo.logspaces` table to obtain undo log file usage information. See the description of this table as well as NDB Cluster Disk Data Objects, for more information.

  The remainder of the discussion in this item applies only to NDB 8.0.12 and earlier. For the row having `NULL` in the `FILE_NAME` column, the value of the `FILE_ID` column is always `0`, that of the `FILE_TYPE` column is always `UNDO LOG`, 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:

```
mysql> 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:

```
mysql> 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;
+--------------+---------+---------+------+----------+
| File         | Free    | Total   | Size | Initial  |
+--------------+---------+---------+------+----------+
| undofile.dat |    NULL | 4194304 |    4 | 16777216 |
| NULL         | 4184068 |    NULL |    4 |     NULL |
+--------------+---------+---------+------+----------+
```

  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:

```
mysql> ALTER LOGFILE GROUP lg1
         ADD UNDOFILE 'undofile02.dat'
         INITIAL_SIZE = 4M
         ENGINE = NDB;
mysql> 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;
+---------------+---------+---------+------+----------+
| File          | Free    | Total   | Size | Initial  |
```

```
+----------------+---------+---------+------+----------+
| undofile.dat   |    NULL | 4194304 |    4 | 16777216 |
| undofile02.dat |    NULL | 1048576 |    4 |  4194304 |
| NULL           | 5223944 |    NULL |    4 |     NULL |
+----------------+---------+---------+------+----------+
```

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:

```
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 |
+--------------+------------+
|      5223944 |   20895776 |
+--------------+------------+
```

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:

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

- Prior to NDB 8.0.13, an additional row was present in the `FILES` table for each NDB Cluster Disk Data tablespace. Because it did not correspond to an actual file, it was removed in NDB 8.0.13. This row had `NULL` for the value of the `FILE_NAME` column, the value of the `FILE_ID` column was always `0`, that of the `FILE_TYPE` column was always `TABLESPACE`, that of the `STATUS` column was always `NORMAL`, and the value of the `ENGINE` column is always `NDBCLUSTER`.

  In NDB 8.0.13 and later, you can obtain information about Disk Data tablespaces using the `ndb_desc` utility. For more information, see NDB Cluster Disk Data Objects, as well as the description of `ndb_desc`.

- For additional information, and examples of creating, dropping, and obtaining information about NDB Cluster Disk Data objects, see NDB Cluster Disk Data Tables.

4.16 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 name t1 and t3 that have the following definitions:

```
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.

4.17 The INFORMATION_SCHEMA KEYWORDS Table

The KEYWORDS table lists the words considered keywords by MySQL and, for each one, indicates whether it is reserved. Reserved keywords may require special treatment in some contexts, such as special quoting when used as identifiers (see Keywords and Reserved Words). This table provides applications a runtime source of MySQL keyword information.

Prior to MySQL 8.0.13, selecting from the KEYWORDS table with no default database selected produced an error. (Bug #90160, Bug #27729859)

The KEYWORDS table has these columns:

- WORD

  The keyword.

- RESERVED

  An integer indicating whether the keyword is reserved (1) or nonreserved (0).

These queries lists all keywords, all reserved keywords, and all nonreserved keywords, respectively:

```
SELECT * FROM INFORMATION_SCHEMA.KEYWORDS;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 1;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED = 0;
```

The latter two queries are equivalent to:

```
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE RESERVED;
SELECT WORD FROM INFORMATION_SCHEMA.KEYWORDS WHERE NOT RESERVED;
```

If you build MySQL from source, the build process generates a `keyword_list.h` header file containing an array of keywords and their reserved status. This file can be found in the `sql` directory under the build directory. This file may be useful for applications that require a static source for the keyword list.

4.18 The INFORMATION_SCHEMA ndb_transid_mysql_connection_map Table

The `ndb_transid_mysql_connection_map` table provides a mapping between `NDB` transactions, `NDB` transaction coordinators, and MySQL Servers attached to an NDB Cluster as API nodes. This information is used when populating the `server_operations` and `server_transactions` tables of the `ndbinfo` NDB Cluster information database.

`INFORMATION_SCHEMA` Name	`SHOW` Name	Remarks
`mysql_connection_id`		MySQL Server connection ID
`node_id`		Transaction coordinator node ID
`ndb_transid`		`NDB` transaction ID

The `mysql_connection_id` is the same as the connection or session ID shown in the output of `SHOW PROCESSLIST`.

There are no `SHOW` statements associated with this table.

This is a nonstandard table, specific to NDB Cluster. It is implemented as an `INFORMATION_SCHEMA` plugin; you can verify that it is supported by checking the output of `SHOW PLUGINS`. If `ndb_transid_mysql_connection_map` support is enabled, the output from this statement includes a plugin having this name, of type `INFORMATION SCHEMA`, and having status `ACTIVE`, as shown here (using emphasized text):

```
mysql> SHOW PLUGINS;
+----------------------------------+--------+--------------------+---------+---------+
| Name                             | Status | Type               | Library | License |
+----------------------------------+--------+--------------------+---------+---------+
| binlog                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| mysql_native_password            | ACTIVE | AUTHENTICATION     | NULL    | GPL     |
| CSV                              | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MEMORY                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MRG_MYISAM                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| MyISAM                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| PERFORMANCE_SCHEMA               | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| BLACKHOLE                        | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ARCHIVE                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbcluster                       | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndbinfo                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| ndb_transid_mysql_connection_map | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| InnoDB                           | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
| INNODB_TRX                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCKS                     | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_LOCK_WAITS                | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP                       | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMP_RESET                 | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM                    | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
| INNODB_CMPMEM_RESET              | ACTIVE | INFORMATION SCHEMA | NULL    | GPL     |
```

```
| partition                          | ACTIVE | STORAGE ENGINE     | NULL    | GPL     |
+------------------------------------+--------+--------------------+---------+---------+
22 rows in set (0.00 sec)
```

The plugin is enabled by default. You can disable it (or force the server not to run unless the plugin starts) by starting the server with the `--ndb-transid-mysql-connection-map` option. If the plugin is disabled, the status is shown by `SHOW PLUGINS` as `DISABLED`. The plugin cannot be enabled or disabled at runtime.

Although the names of this table and its columns are displayed using lowercase, you can use uppercase or lowercase when referring to them in SQL statements.

For this table to be created, the MySQL Server must be a binary supplied with the NDB Cluster distribution, or one built from the NDB Cluster sources with `NDB` storage engine support enabled. It is not available in the standard MySQL 8.0 Server.

4.19 The INFORMATION_SCHEMA OPTIMIZER_TRACE Table

The `OPTIMIZER_TRACE` table provides information produced by the optimizer tracing capability for traced statements. To enable tracking, use the `optimizer_trace` system variable. For details, see MySQL Internals: Tracing the Optimizer.

The `OPTIMIZER_TRACE` table has these columns:

- `QUERY`

  The text of the traced statement.

- `TRACE`

  The trace, in `JSON` format.

- `MISSING_BYTES_BEYOND_MAX_MEM_SIZE`

  Each remembered trace is a string that is extended as optimization progresses and appends data to it. The `optimizer_trace_max_mem_size` variable sets a limit on the total amount of memory used by all currently remembered traces. If this limit is reached, the current trace is not extended (and thus is incomplete), and the `MISSING_BYTES_BEYOND_MAX_MEM_SIZE` column shows the number of bytes missing from the trace.

- `INSUFFICIENT_PRIVILEGES`

  If a traced query uses views or stored routines that have `SQL SECURITY` with a value of `DEFINER`, it may be that a user other than the definer is denied from seeing the trace of the query. In that case, the trace is shown as empty and `INSUFFICIENT_PRIVILEGES` has a value of 1. Otherwise, the value is 0.

4.20 The INFORMATION_SCHEMA PARAMETERS Table

The `PARAMETERS` table provides information about parameters for stored routines (stored procedures and stored functions), and about return values for stored functions. The `PARAMETERS` table does not include built-in (native) functions or loadable functions.

The `PARAMETERS` table has these columns:

- `SPECIFIC_CATALOG`

The name of the catalog to which the routine containing the parameter belongs. This value is always `def`.

- `SPECIFIC_SCHEMA`

  The name of the schema (database) to which the routine containing the parameter belongs.

- `SPECIFIC_NAME`

  The name of the routine containing the parameter.

- `ORDINAL_POSITION`

  For successive parameters of a stored procedure or function, the `ORDINAL_POSITION` values are 1, 2, 3, and so forth. For a stored function, there is also a row that applies to the function return value (as described by the `RETURNS` clause). The return value is not a true parameter, so the row that describes it has these unique characteristics:

  - The `ORDINAL_POSITION` value is 0.
  - The `PARAMETER_NAME` and `PARAMETER_MODE` values are `NULL` because the return value has no name and the mode does not apply.

- `PARAMETER_MODE`

  The mode of the parameter. This value is one of `IN`, `OUT`, or `INOUT`. For a stored function return value, this value is `NULL`.

- `PARAMETER_NAME`

  The name of the parameter. For a stored function return value, this value is `NULL`.

- `DATA_TYPE`

  The parameter data type.

  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 string parameters, the maximum length in characters.

- `CHARACTER_OCTET_LENGTH`

  For string parameters, the maximum length in bytes.

- `NUMERIC_PRECISION`

  For numeric parameters, the numeric precision.

- `NUMERIC_SCALE`

  For numeric parameters, the numeric scale.

- `DATETIME_PRECISION`

  For temporal parameters, the fractional seconds precision.

- `CHARACTER_SET_NAME`

  For character string parameters, the character set name.

- `COLLATION_NAME`

  For character string parameters, the collation name.

- `DTD_IDENTIFIER`

  The parameter data type.

  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_TYPE`

  `PROCEDURE` for stored procedures, `FUNCTION` for stored functions.

4.21 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:

```
CREATE TABLE tp (
    c1 INT,
    c2 INT,
    c3 VARCHAR(25)
)
PARTITION BY HASH(c1 + c2)
PARTITIONS 4;
```

  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

- PARTITIONS 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, 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.

4.22 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

- `PLUGINS` 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:

```
SELECT
  PLUGIN_NAME, PLUGIN_STATUS, PLUGIN_TYPE,
  PLUGIN_LIBRARY, PLUGIN_LICENSE
FROM INFORMATION_SCHEMA.PLUGINS;
SHOW PLUGINS;
```

4.23 The INFORMATION_SCHEMA PROCESSLIST Table

The MySQL process list indicates the operations currently being performed by the set of threads executing within the server. The `PROCESSLIST` table is one source of process information. For a comparison of this table with other sources, see Sources of Process Information.

The `PROCESSLIST` table has these columns:

- `ID`

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

- `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, for example, a delayed-row handler thread or an I/O or SQL thread used on replica hosts. 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 occurred. `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 for the thread, or `NULL` if none has been selected.

- `COMMAND`

  The type of command the thread is executing on behalf of the client, or `Sleep` if the session is idle. For descriptions of thread commands, see Examining Server Thread (Process) 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 replica SQL thread, the value is the number of seconds between the timestamp of the last replicated event and the real time of the replica host. See Replication Threads.

- `STATE`

  An action, event, or state that indicates what the thread is doing. For descriptions of `STATE` values, see Examining Server Thread (Process) 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.

- `INFO`

  The statement the thread is executing, or `NULL` if it is executing no 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

- PROCESSLIST is a nonstandard INFORMATION_SCHEMA table.
- Like the output from the SHOW PROCESSLIST statement, the PROCESSLIST table provides information about all threads, even those belonging to other users, if you have the PROCESS privilege. Otherwise (without the PROCESS privilege), nonanonymous users have access to information about their own threads but not threads for other users, and anonymous users have no access to thread information.
- 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.

The following statements are equivalent:

```
SELECT * FROM INFORMATION_SCHEMA.PROCESSLIST
SHOW FULL PROCESSLIST
```

4.24 The INFORMATION_SCHEMA PROFILING Table

The PROFILING table provides statement profiling information. Its contents correspond to the information produced by the SHOW PROFILE and SHOW PROFILES statements (see SHOW PROFILE Statement). The table is empty unless the profiling session variable is set to 1.

> **Note**
>
> This table is deprecated; expect it to be removed in a future MySQL release. Use the Performance Schema instead; see Query Profiling Using Performance Schema.

The PROFILING table has these columns:

- QUERY_ID

  A numeric statement identifier.

- SEQ

  A sequence number indicating the display order for rows with the same QUERY_ID value.

- STATE

  The profiling state to which the row measurements apply.

- DURATION

  How long statement execution remained in the given state, in seconds.

- CPU_USER, CPU_SYSTEM

  User and system CPU use, in seconds.

- CONTEXT_VOLUNTARY, CONTEXT_INVOLUNTARY

  How many voluntary and involuntary context switches occurred.

- BLOCK_OPS_IN, BLOCK_OPS_OUT

  The number of block input and output operations.

- `MESSAGES_SENT`, `MESSAGES_RECEIVED`

  The number of communication messages sent and received.

- `PAGE_FAULTS_MAJOR`, `PAGE_FAULTS_MINOR`

  The number of major and minor page faults.

- `SWAPS`

  How many swaps occurred.

- `SOURCE_FUNCTION`, `SOURCE_FILE`, and `SOURCE_LINE`

  Information indicating where in the source code the profiled state executes.

Notes

- `PROFILING` is a nonstandard `INFORMATION_SCHEMA` table.

Profiling information is also available from the `SHOW PROFILE` and `SHOW PROFILES` statements. See SHOW PROFILE Statement. For example, the following queries are equivalent:

```
SHOW PROFILE FOR QUERY 2;
SELECT STATE, FORMAT(DURATION, 6) AS DURATION
FROM INFORMATION_SCHEMA.PROFILING
WHERE QUERY_ID = 2 ORDER BY SEQ;
```

4.25 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.

4.26 The INFORMATION_SCHEMA RESOURCE_GROUPS Table

The `RESOURCE_GROUPS` table provides access to information about resource groups. For general discussion of the resource group capability, see Resource Groups.

You can see information only for columns for which you have some privilege.

The `RESOURCE_GROUPS` table has these columns:

- `RESOURCE_GROUP_NAME`

  The name of the resource group.

- `RESOURCE_GROUP_TYPE`

  The resource group type, either `SYSTEM` or `USER`.

- `RESOURCE_GROUP_ENABLED`

  Whether the resource group is enabled (1) or disabled (0);

- `VCPU_IDS`

  The CPU affinity; that is, the set of virtual CPUs that the resource group can use. The value is a list of comma-separated CPU numbers or ranges.

- `THREAD_PRIORITY`

  The priority for threads assigned to the resource group. The priority ranges from -20 (highest priority) to 19 (lowest priority). System resource groups have a priority that ranges from -20 to 0. User resource groups have a priority that ranges from 0 to 19.

4.27 The INFORMATION_SCHEMA ROLE_COLUMN_GRANTS Table

The `ROLE_COLUMN_GRANTS` table (available as of MySQL 8.0.19) provides information about the column privileges for roles that are available to or granted by the currently enabled roles.

The ROLE_COLUMN_GRANTS table has these columns:

- GRANTOR

  The user name part of the account that granted the role.

- GRANTOR_HOST

  The host name part of the account that granted the role.

- GRANTEE

  The user name part of the account to which the role is granted.

- GRANTEE_HOST

  The host name part of the account to which the role is granted.

- TABLE_CATALOG

  The name of the catalog to which the role applies. This value is always def.

- TABLE_SCHEMA

  The name of the schema (database) to which the role applies.

- TABLE_NAME

  The name of the table to which the role applies.

- COLUMN_NAME

  The name of the column to which the role applies.

- 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.

- IS_GRANTABLE

  YES or NO, depending on whether the role is grantable to other accounts.

4.28 The INFORMATION_SCHEMA ROLE_ROUTINE_GRANTS Table

The ROLE_ROUTINE_GRANTS table (available as of MySQL 8.0.19) provides information about the routine privileges for roles that are available to or granted by the currently enabled roles.

The ROLE_ROUTINE_GRANTS table has these columns:

- GRANTOR

  The user name part of the account that granted the role.

- GRANTOR_HOST

  The host name part of the account that granted the role.

- `GRANTEE`

  The user name part of the account to which the role is granted.

- `GRANTEE_HOST`

  The host name part of the account to which the role is granted.

- `SPECIFIC_CATALOG`

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

- `SPECIFIC_SCHEMA`

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

- `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.

- `PRIVILEGE_TYPE`

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

- `IS_GRANTABLE`

  `YES` or `NO`, depending on whether the role is grantable to other accounts.

4.29 The INFORMATION_SCHEMA ROLE_TABLE_GRANTS Table

The `ROLE_TABLE_GRANTS` table (available as of MySQL 8.0.19) provides information about the table privileges for roles that are available to or granted by the currently enabled roles.

The `ROLE_TABLE_GRANTS` table has these columns:

- `GRANTOR`

  The user name part of the account that granted the role.

- `GRANTOR_HOST`

  The host name part of the account that granted the role.

- `GRANTEE`

The user name part of the account to which the role is granted.

- `GRANTEE_HOST`

  The host name part of the account to which the role is granted.

- `TABLE_CATALOG`

  The name of the catalog to which the role applies. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the role applies.

- `TABLE_NAME`

  The name of the table to which the role applies.

- `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 column privilege held by the grantee.

- `IS_GRANTABLE`

  `YES` or `NO`, depending on whether the role is grantable to other accounts.

4.30 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 (native) functions or loadable functions.

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.

4.31 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:

```
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.

Notes

- The SCHEMATA_EXTENSIONS table augments the SCHEMATA table with information about schema options.

4.32 The INFORMATION_SCHEMA SCHEMATA_EXTENSIONS Table

The SCHEMATA_EXTENSIONS table (available as of MySQL 8.0.22) augments the SCHEMATA table with information about schema options.

The SCHEMATA_EXTENSIONS 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.

- OPTIONS

  The options for the schema. If the schema is read only, the value contains READ ONLY=1. If the schema is not read only, no READ ONLY option appears.

Example

```
mysql> ALTER SCHEMA mydb READ ONLY = 1;
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA_EXTENSIONS
       WHERE SCHEMA_NAME = 'mydb';
+--------------+-------------+-------------+
| CATALOG_NAME | SCHEMA_NAME | OPTIONS     |
+--------------+-------------+-------------+
| def          | mydb        | READ ONLY=1 |
+--------------+-------------+-------------+
mysql> ALTER SCHEMA mydb READ ONLY = 0;
mysql> SELECT * FROM INFORMATION_SCHEMA.SCHEMATA_EXTENSIONS
       WHERE SCHEMA_NAME = 'mydb';
+--------------+-------------+---------+
| CATALOG_NAME | SCHEMA_NAME | OPTIONS |
+--------------+-------------+---------+
| def          | mydb        |         |
+--------------+-------------+---------+
```

Notes

- SCHEMATA_EXTENSIONS is a nonstandard INFORMATION_SCHEMA table.

4.33 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

- SCHEMA_PRIVILEGES is a nonstandard INFORMATION_SCHEMA table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.SCHEMA_PRIVILEGES
SHOW GRANTS ...
```

4.34 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:

```
SELECT * FROM INFORMATION_SCHEMA.STATISTICS
  WHERE table_name = 'tbl_name'
  AND table_schema = 'db_name'
SHOW INDEX
  FROM tbl_name
  FROM db_name
```

4.35 The INFORMATION_SCHEMA ST_GEOMETRY_COLUMNS Table

The `ST_GEOMETRY_COLUMNS` table provides information about table columns that store spatial data. This table is based on the SQL/MM (ISO/IEC 13249-3) standard, with extensions as noted. MySQL implements `ST_GEOMETRY_COLUMNS` as a view on the `INFORMATION_SCHEMA COLUMNS` table.

The `ST_GEOMETRY_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.

- `SRS_NAME`

  The spatial reference system (SRS) name.

- `SRS_ID`

  The spatial reference system ID (SRID).

- `GEOMETRY_TYPE_NAME`

  The column data type. Permitted values are: `geometry`, `point`, `linestring`, `polygon`, `multipoint`, `multilinestring`, `multipolygon`, `geometrycollection`. This column is a MySQL extension to the standard.

4.36 The INFORMATION_SCHEMA ST_SPATIAL_REFERENCE_SYSTEMS Table

The `ST_SPATIAL_REFERENCE_SYSTEMS` table provides information about available spatial reference systems (SRSs) for spatial data. This table is based on the SQL/MM (ISO/IEC 13249-3) standard.

Entries in the `ST_SPATIAL_REFERENCE_SYSTEMS` table are based on the European Petroleum Survey Group (EPSG) data set, except for SRID 0, which corresponds to a special SRS used in MySQL that represents an infinite flat Cartesian plane with no units assigned to its axes. For additional information about SRSs, see Spatial Reference System Support.

The `ST_SPATIAL_REFERENCE_SYSTEMS` table has these columns:

- `SRS_NAME`

  The spatial reference system name. This value is unique.

- `SRS_ID`

  The spatial reference system numeric ID. This value is unique.

  `SRS_ID` values represent the same kind of values as the SRID of geometry values or passed as the SRID argument to spatial functions. SRID 0 (the unitless Cartesian plane) is special. It is always a legal spatial reference system ID and can be used in any computations on spatial data that depend on SRID values.

- `ORGANIZATION`

  The name of the organization that defined the coordinate system on which the spatial reference system is based.

- `ORGANIZATION_COORDSYS_ID`

  The numeric ID given to the spatial reference system by the organization that defined it.

- `DEFINITION`

  The spatial reference system definition. `DEFINITION` values are WKT values, represented as specified in the Open Geospatial Consortium document OGC 12-063r5.

  SRS definition parsing occurs on demand when definitions are needed by GIS functions. Parsed definitions are stored in the data dictionary cache to enable reuse and avoid incurring parsing overhead for every statement that needs SRS information.

- `DESCRIPTION`

  The spatial reference system description.

Notes

- The `SRS_NAME`, `ORGANIZATION`, `ORGANIZATION_COORDSYS_ID`, and `DESCRIPTION` columns contain information that may be of interest to users, but they are not used by MySQL.

Example

```
mysql> SELECT * FROM ST_SPATIAL_REFERENCE_SYSTEMS
       WHERE SRS_ID = 4326\G
*************************** 1. row ***************************
                SRS_NAME: WGS 84
                  SRS_ID: 4326
            ORGANIZATION: EPSG
ORGANIZATION_COORDSYS_ID: 4326
              DEFINITION: GEOGCS["WGS 84",DATUM["World Geodetic System 1984",
                          SPHEROID["WGS 84",6378137,298.257223563,
                          AUTHORITY["EPSG","7030"]],AUTHORITY["EPSG","6326"]],
                          PRIMEM["Greenwich",0,AUTHORITY["EPSG","8901"]],
                          UNIT["degree",0.017453292519943278,
                          AUTHORITY["EPSG","9122"]],
                          AXIS["Lat",NORTH],AXIS["Long",EAST],
                          AUTHORITY["EPSG","4326"]]
             DESCRIPTION:
```

This entry describes the SRS used for GPS systems. It has a name (`SRS_NAME`) of WGS 84 and an ID (`SRS_ID`) of 4326, which is the ID used by the European Petroleum Survey Group (EPSG).

The `DEFINITION` values for projected and geographic SRSs begin with `PROJCS` and `GEOGCS`, respectively. The definition for SRID 0 is special and has an empty `DEFINITION` value. The following query determines how many entries in the `ST_SPATIAL_REFERENCE_SYSTEMS` table correspond to projected, geographic, and other SRSs, based on `DEFINITION` values:

```
mysql> SELECT
         COUNT(*),
         CASE LEFT(DEFINITION, 6)
           WHEN 'PROJCS' THEN 'Projected'
           WHEN 'GEOGCS' THEN 'Geographic'
           ELSE 'Other'
         END AS SRS_TYPE
```

```
      FROM INFORMATION_SCHEMA.ST_SPATIAL_REFERENCE_SYSTEMS
      GROUP BY SRS_TYPE;
+----------+------------+
| COUNT(*) | SRS_TYPE   |
+----------+------------+
|        1 | Other      |
|     4668 | Projected  |
|      483 | Geographic |
+----------+------------+
```

To enable manipulation of SRS entries stored in the data dictionary, MySQL provides these SQL statements:

- `CREATE SPATIAL REFERENCE SYSTEM`: See CREATE SPATIAL REFERENCE SYSTEM Statement. The description for this statement includes additional information about SRS components.
- `DROP SPATIAL REFERENCE SYSTEM`: See DROP SPATIAL REFERENCE SYSTEM Statement.

4.37 The INFORMATION_SCHEMA ST_UNITS_OF_MEASURE Table

The `ST_UNITS_OF_MEASURE` table (available as of MySQL 8.0.14) provides information about acceptable units for the `ST_Distance()` function.

The `ST_UNITS_OF_MEASURE` table has these columns:

- `UNIT_NAME`

  The name of the unit.

- `UNIT_TYPE`

  The unit type (for example, `LINEAR`).

- `CONVERSION_FACTOR`

  A conversion factor used for internal calculations.

- `DESCRIPTION`

  A description of the unit.

4.38 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 Section 4.21, “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.

  CREATE_OPTIONS shows partitioned for a partitioned table.

  Prior to MySQL 8.0.16, CREATE_OPTIONS shows the ENCRYPTION clause specified for tables created in file-per-table tablespaces. As of MySQL 8.0.16, it 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 created in general tablespaces. To identify encrypted file-per-table and general tablespaces, query the INNODB_TABLESPACES ENCRYPTION column.

  When creating a table with strict mode disabled, the storage engine's default row format is used if the specified row format is not supported. The actual row format of the table is reported in the ROW_FORMAT column. CREATE_OPTIONS shows the row format that was specified in the CREATE TABLE statement.

  When altering the storage engine of a table, table options that are not applicable to the new storage engine are retained in the table definition to enable reverting the table with its previously defined options to the original storage engine, if necessary. The CREATE_OPTIONS column may show retained options.

- 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:

```
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:

```
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']
```

4.39 The INFORMATION_SCHEMA TABLES_EXTENSIONS Table

The `TABLES_EXTENSIONS` table (available as of MySQL 8.0.21) provides information about table attributes defined for primary and secondary storage engines.

> **Note**
>
> The `TABLES_EXTENSIONS` table is reserved for future use.

The `TABLES_EXTENSIONS` 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.

- ENGINE_ATTRIBUTE

  Table attributes defined for the primary storage engine. Reserved for future use.

- SECONDARY_ENGINE_ATTRIBUTE

  Table attributes defined for the secondary storage engine. Reserved for future use.

4.40 The INFORMATION_SCHEMA TABLESPACES Table

This table is unused. It is deprecated; expect it to be removed in a future MySQL release. Other INFORMATION_SCHEMA tables may provide related information:

- For NDB, the INFORMATION_SCHEMA FILES table provides tablespace-related information.
- For InnoDB, the INFORMATION_SCHEMA INNODB_TABLESPACES and INNODB_DATAFILES tables provide tablespace metadata.

4.41 The INFORMATION_SCHEMA TABLESPACES_EXTENSIONS Table

The TABLESPACES_EXTENSIONS table (available as of MySQL 8.0.21) provides information about tablespace attributes defined for primary storage engines.

> **Note**
>
> The TABLESPACES_EXTENSIONS table is reserved for future use.

The TABLESPACES_EXTENSIONS table has these columns:

- TABLESPACE_NAME

  The name of the tablespace.

- ENGINE_ATTRIBUTE

  Tablespace attributes defined for the primary storage engine. Reserved for future use.

4.42 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.

- The `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.

4.43 The INFORMATION_SCHEMA TABLE_CONSTRAINTS_EXTENSIONS Table

The `TABLE_CONSTRAINTS_EXTENSIONS` table (available as of MySQL 8.0.21) provides information about table constraint attributes defined for primary and secondary storage engines.

> **Note**
>
> The `TABLE_CONSTRAINTS_EXTENSIONS` table is reserved for future use.

The `TABLE_CONSTRAINTS_EXTENSIONS` table has these columns:

- `CONSTRAINT_CATALOG`

  The name of the catalog to which the table belongs.

- `CONSTRAINT_SCHEMA`

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

- `CONSTRAINT_NAME`

  The name of the constraint.

- `TABLE_NAME`

  The name of the table.

- `ENGINE_ATTRIBUTE`

  Constraint attributes defined for the primary storage engine. Reserved for future use.

- `SECONDARY_ENGINE_ATTRIBUTE`

  Constraint attributes defined for the secondary storage engine. Reserved for future use.

4.44 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

- TABLE_PRIVILEGES is a nonstandard INFORMATION_SCHEMA table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.TABLE_PRIVILEGES
SHOW GRANTS ...
```

4.45 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:

```
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,
                           ERROR_FOR_DIVISION_BY_ZERO,
                           NO_ENGINE_SUBSTITUTION
                  DEFINER: me@localhost
      CHARACTER_SET_CLIENT: utf8mb4
      COLLATION_CONNECTION: utf8mb4_0900_ai_ci
        DATABASE_COLLATION: utf8mb4_0900_ai_ci
```

Trigger information is also available from the SHOW TRIGGERS statement. See SHOW TRIGGERS Statement.

4.46 The INFORMATION_SCHEMA USER_ATTRIBUTES Table

The USER_ATTRIBUTES table (available as of MySQL 8.0.21) provides information about user comments and user attributes. It takes its values from the mysql.user system table.

The USER_ATTRIBUTES table has these columns:

- USER

  The user name portion of the account to which the ATTRIBUTE column value applies.

- HOST

  The host name portion of the account to which the ATTRIBUTE column value applies.

- ATTRIBUTE

  The user comment, user attribute, or both belonging to the account specified by the USER and HOST columns. The value is in JSON object notation. Attributes are shown exactly as set using CREATE USER and ALTER USER statements with ATTRIBUTE or COMMENT options. A comment is shown as a key-value pair having comment as the key. For additional information and examples, see CREATE USER Comment and Attribute Options.

Notes

- USER_ATTRIBUTES is a nonstandard INFORMATION_SCHEMA table.
- To obtain only the user comment for a given user as an unquoted string, you can employ a query such as this one:

```
mysql> SELECT ATTRIBUTE->>"$.comment" AS Comment
    ->     FROM INFORMATION_SCHEMA.USER_ATTRIBUTES
    ->     WHERE USER='bill' AND HOST='localhost';
+-----------+
| Comment   |
+-----------+
| A comment |
+-----------+
```

  Similarly, you can obtain the unquoted value for a given user attribute using its key.

- Prior to MySQL 8.0.22, USER_ATTRIBUTES contents are accessible by anyone. As of MySQL 8.0.22, USER_ATTRIBUTES contents are accessible as follows:
  - All rows are accessible if:
    - The current thread is a replica thread.
    - The access control system has not been initialized (for example, the server was started with the --skip-grant-tables option).
    - The currently authenticated account has the UPDATE or SELECT privilege for the mysql.user system table.
    - The currently authenticated account has the CREATE USER and SYSTEM_USER privileges.
  - Otherwise, the currently authenticated account can see the row for that account. Additionally, if the account has the CREATE USER privilege but not the SYSTEM_USER privilege, it can see rows for all other accounts that do not have the SYSTEM_USER privilege.

For more information about specifying account comments and attributes, see CREATE USER Statement.

4.47 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

- USER_PRIVILEGES is a nonstandard INFORMATION_SCHEMA table.

The following statements are *not* equivalent:

```
SELECT ... FROM INFORMATION_SCHEMA.USER_PRIVILEGES
SHOW GRANTS ...
```

4.48 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:

```
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:

```
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.

4.49 The INFORMATION_SCHEMA VIEW_ROUTINE_USAGE Table

The `VIEW_ROUTINE_USAGE` table (available as of MySQL 8.0.13) provides access to information about stored functions used in view definitions. The table does not list information about built-in (native) functions or loadable functions used in the definitions.

You can see information only for views for which you have some privilege, and only for functions for which you have some privilege.

The `VIEW_ROUTINE_USAGE` 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.

- `SPECIFIC_CATALOG`

  The name of the catalog to which the function used in the view definition belongs. This value is always `def`.

- `SPECIFIC_SCHEMA`

  The name of the schema (database) to which the function used in the view definition belongs.

- `SPECIFIC_NAME`

  The name of the function used in the view definition.

4.50 The INFORMATION_SCHEMA VIEW_TABLE_USAGE Table

The `VIEW_TABLE_USAGE` table (available as of MySQL 8.0.13) provides access to information about tables and views used in view definitions.

You can see information only for views for which you have some privilege, and only for tables for which you have some privilege.

The `VIEW_TABLE_USAGE` table has these columns:

- `VIEW_CATALOG`

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

- `VIEW_SCHEMA`

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

- `VIEW_NAME`

  The name of the view.

- `TABLE_CATALOG`

  The name of the catalog to which the table or view used in the view definition belongs. This value is always `def`.

- `TABLE_SCHEMA`

  The name of the schema (database) to which the table or view used in the view definition belongs.

- `TABLE_NAME`

  The name of the table or view used in the view definition.

Chapter 5 INFORMATION_SCHEMA InnoDB Tables

Table of Contents

This section provides table definitions for `INFORMATION_SCHEMA InnoDB` tables. For related information and examples, see InnoDB INFORMATION_SCHEMA Tables.

`INFORMATION_SCHEMA InnoDB` tables can be used to monitor ongoing `InnoDB` activity, to detect inefficiencies before they turn into issues, or to troubleshoot performance and capacity issues. As your database becomes bigger and busier, running up against the limits of your hardware capacity, you monitor and tune these aspects to keep the database running smoothly.

5.1 INFORMATION_SCHEMA InnoDB Table Reference

The following table summarizes `INFORMATION_SCHEMA` InnoDB tables. For greater detail, see the individual table descriptions.

Table 5.1 INFORMATION_SCHEMA InnoDB Tables

Table Name	Description	Introduced
`INNODB_BUFFER_PAGE`	Pages in InnoDB buffer pool	

Table Name	Description	Introduced
INNODB_BUFFER_PAGE_LRU	LRU ordering of pages in InnoDB buffer pool	
INNODB_BUFFER_POOL_STATS	InnoDB buffer pool statistics	
INNODB_CACHED_INDEXES	Number of index pages cached per index in InnoDB buffer pool	
INNODB_CMP	Status for operations related to compressed InnoDB tables	
INNODB_CMP_PER_INDEX	Status for operations related to compressed InnoDB tables and indexes	
INNODB_CMP_PER_INDEX_RESET	Status for operations related to compressed InnoDB tables and indexes	
INNODB_CMP_RESET	Status for operations related to compressed InnoDB tables	
INNODB_CMPMEM	Status for compressed pages within InnoDB buffer pool	
INNODB_CMPMEM_RESET	Status for compressed pages within InnoDB buffer pool	
INNODB_COLUMNS	Columns in each InnoDB table	
INNODB_DATAFILES	Data file path information for InnoDB file-per-table and general tablespaces	
INNODB_FIELDS	Key columns of InnoDB indexes	
INNODB_FOREIGN	InnoDB foreign-key metadata	
INNODB_FOREIGN_COLS	InnoDB foreign-key column status information	
INNODB_FT_BEING_DELETED	Snapshot of INNODB_FT_DELETED table	
INNODB_FT_CONFIG	Metadata for InnoDB table FULLTEXT index and associated processing	
INNODB_FT_DEFAULT_STOPWORD	Default list of stopwords for InnoDB FULLTEXT indexes	
INNODB_FT_DELETED	Rows deleted from InnoDB table FULLTEXT index	
INNODB_FT_INDEX_CACHE	Token information for newly inserted rows in InnoDB FULLTEXT index	
INNODB_FT_INDEX_TABLE	Inverted index information for processing text searches against InnoDB table FULLTEXT index	
INNODB_INDEXES	InnoDB index metadata	
INNODB_METRICS	InnoDB performance information	

Table Name	Description	Introduced
INNODB_SESSION_TEMP_TABLESPACES	Session temporary-tablespace metadata	8.0.13
INNODB_TABLES	InnoDB table metadata	
INNODB_TABLESPACES	InnoDB file-per-table, general, and undo tablespace metadata	
INNODB_TABLESPACES_BRIEF	Brief file-per-table, general, undo, and system tablespace metadata	
INNODB_TABLESTATS	InnoDB table low-level status information	
INNODB_TEMP_TABLE_INFO	Information about active user-created InnoDB temporary tables	
INNODB_TRX	Active InnoDB transaction information	
INNODB_VIRTUAL	InnoDB virtual generated column metadata	

5.2 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE Table

The INNODB_BUFFER_PAGE table provides information about each page in the InnoDB buffer pool.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Buffer Pool Tables.

> **Warning**
>
> Querying the INNODB_BUFFER_PAGE table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The INNODB_BUFFER_PAGE table has these columns:

- POOL_ID

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- BLOCK_ID

  The buffer pool block ID.

- SPACE

  The tablespace ID; the same value as INNODB_TABLES.SPACE.

- PAGE_NUMBER

  The page number.

- PAGE_TYPE

  The page type. The following table shows the permitted values.

Table 5.2 INNODB_BUFFER_PAGE.PAGE_TYPE Values

Page Type	Description
ALLOCATED	Freshly allocated page
BLOB	Uncompressed BLOB page
COMPRESSED_BLOB2	Subsequent comp BLOB page
COMPRESSED_BLOB	First compressed BLOB page
ENCRYPTED_RTREE	Encrypted R-tree
EXTENT_DESCRIPTOR	Extent descriptor page
FILE_SPACE_HEADER	File space header
FIL_PAGE_TYPE_UNUSED	Unused
IBUF_BITMAP	Insert buffer bitmap
IBUF_FREE_LIST	Insert buffer free list
IBUF_INDEX	Insert buffer index
INDEX	B-tree node
INODE	Index node
LOB_DATA	Uncompressed LOB data
LOB_FIRST	First page of uncompressed LOB
LOB_INDEX	Uncompressed LOB index
PAGE_IO_COMPRESSED	Compressed page
PAGE_IO_COMPRESSED_ENCRYPTED	Compressed and encrypted page
PAGE_IO_ENCRYPTED	Encrypted page
RSEG_ARRAY	Rollback segment array
RTREE_INDEX	R-tree index
SDI_BLOB	Uncompressed SDI BLOB
SDI_COMPRESSED_BLOB	Compressed SDI BLOB
SDI_INDEX	SDI index
SYSTEM	System page
TRX_SYSTEM	Transaction system data
UNDO_LOG	Undo log page
UNKNOWN	Unknown
ZLOB_DATA	Compressed LOB data
ZLOB_FIRST	First page of compressed LOB
ZLOB_FRAG	Compressed LOB fragment
ZLOB_FRAG_ENTRY	Compressed LOB fragment index
ZLOB_INDEX	Compressed LOB index

- FLUSH_TYPE

  The flush type.

- `FIX_COUNT`

  The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- `IS_HASHED`

  Whether a hash index has been built on this page.

- `NEWEST_MODIFICATION`

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`. The column is `NULL` if the server has not yet accessed the table.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `PAGE_STATE`

  The page state. The following table shows the permitted values.

  Table 5.3 INNODB_BUFFER_PAGE.PAGE_STATE Values

Page State	Description
`FILE_PAGE`	A buffered file page
`MEMORY`	Contains a main memory object
`NOT_USED`	In the free list
`NULL`	Clean compressed pages, compressed pages in the flush list, pages used as buffer pool watch sentinels

Page State	Description
READY_FOR_USE	A free page
REMOVE_HASH	Hash index should be removed before placing in the free list

- IO_FIX

  Whether any I/O is pending for this page: IO_NONE = no pending I/O, IO_READ = read pending, IO_WRITE = write pending, IO_PIN = relocation and removal from the flush not permitted.

- IS_OLD

  Whether the block is in the sublist of old blocks in the LRU list.

- FREE_PAGE_CLOCK

  The value of the freed_page_clock counter when the block was the last placed at the head of the LRU list. The freed_page_clock counter tracks the number of blocks removed from the end of the LRU list.

- IS_STALE

  Whether the page is stale. Added in MySQL 8.0.24.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE LIMIT 1\G
*************************** 1. row ***************************
           POOL_ID: 0
         BLOCK_ID: 0
            SPACE: 97
      PAGE_NUMBER: 2473
        PAGE_TYPE: INDEX
       FLUSH_TYPE: 1
        FIX_COUNT: 0
        IS_HASHED: YES
NEWEST_MODIFICATION: 733855581
OLDEST_MODIFICATION: 0
       ACCESS_TIME: 3378385672
        TABLE_NAME: `employees`.`salaries`
        INDEX_NAME: PRIMARY
    NUMBER_RECORDS: 468
         DATA_SIZE: 14976
   COMPRESSED_SIZE: 0
        PAGE_STATE: FILE_PAGE
            IO_FIX: IO_NONE
            IS_OLD: YES
   FREE_PAGE_CLOCK: 66
          IS_STALE: NO
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The `INNODB_BUFFER_PAGE` table reports information about these pages until they are evicted from the buffer pool. For more information about how the `InnoDB` manages buffer pool data, see Buffer Pool.

5.3 The INFORMATION_SCHEMA INNODB_BUFFER_PAGE_LRU Table

The `INNODB_BUFFER_PAGE_LRU` table provides information about the pages in the `InnoDB` buffer pool; in particular, how they are ordered in the LRU list that determines which pages to evict from the buffer pool when it becomes full.

The `INNODB_BUFFER_PAGE_LRU` table has the same columns as the `INNODB_BUFFER_PAGE` table with a few exceptions. It has `LRU_POSITION` and `COMPRESSED` columns instead of `BLOCK_ID` and `PAGE_STATE` columns, and it does not include and `IS_STALE` column.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Buffer Pool Tables.

> **Warning**
>
> Querying the `INNODB_BUFFER_PAGE_LRU` table can affect performance. Do not query this table on a production system unless you are aware of the performance impact and have determined it to be acceptable. To avoid impacting performance on a production system, reproduce the issue you want to investigate and query buffer pool statistics on a test instance.

The `INNODB_BUFFER_PAGE_LRU` table has these columns:

- `POOL_ID`

  The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `LRU_POSITION`

  The position of the page in the LRU list.

- `SPACE`

  The tablespace ID; the same value as `INNODB_TABLES.SPACE`.

- `PAGE_NUMBER`

  The page number.

- `PAGE_TYPE`

  The page type. The following table shows the permitted values.

  Table 5.4 INNODB_BUFFER_PAGE_LRU.PAGE_TYPE Values

Page Type	Description
`ALLOCATED`	Freshly allocated page
`BLOB`	Uncompressed BLOB page
`COMPRESSED_BLOB2`	Subsequent comp BLOB page
`COMPRESSED_BLOB`	First compressed BLOB page

Page Type	Description
ENCRYPTED_RTREE	Encrypted R-tree
EXTENT_DESCRIPTOR	Extent descriptor page
FILE_SPACE_HEADER	File space header
FIL_PAGE_TYPE_UNUSED	Unused
IBUF_BITMAP	Insert buffer bitmap
IBUF_FREE_LIST	Insert buffer free list
IBUF_INDEX	Insert buffer index
INDEX	B-tree node
INODE	Index node
LOB_DATA	Uncompressed LOB data
LOB_FIRST	First page of uncompressed LOB
LOB_INDEX	Uncompressed LOB index
PAGE_IO_COMPRESSED	Compressed page
PAGE_IO_COMPRESSED_ENCRYPTED	Compressed and encrypted page
PAGE_IO_ENCRYPTED	Encrypted page
RSEG_ARRAY	Rollback segment array
RTREE_INDEX	R-tree index
SDI_BLOB	Uncompressed SDI BLOB
SDI_COMPRESSED_BLOB	Compressed SDI BLOB
SDI_INDEX	SDI index
SYSTEM	System page
TRX_SYSTEM	Transaction system data
UNDO_LOG	Undo log page
UNKNOWN	Unknown
ZLOB_DATA	Compressed LOB data
ZLOB_FIRST	First page of compressed LOB
ZLOB_FRAG	Compressed LOB fragment
ZLOB_FRAG_ENTRY	Compressed LOB fragment index
ZLOB_INDEX	Compressed LOB index

- FLUSH_TYPE

  The flush type.

- FIX_COUNT

  The number of threads using this block within the buffer pool. When zero, the block is eligible to be evicted.

- IS_HASHED

  Whether a hash index has been built on this page.

- `NEWEST_MODIFICATION`

  The Log Sequence Number of the youngest modification.

- `OLDEST_MODIFICATION`

  The Log Sequence Number of the oldest modification.

- `ACCESS_TIME`

  An abstract number used to judge the first access time of the page.

- `TABLE_NAME`

  The name of the table the page belongs to. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`. The column is `NULL` if the server has not yet accessed the table.

- `INDEX_NAME`

  The name of the index the page belongs to. This can be the name of a clustered index or a secondary index. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `NUMBER_RECORDS`

  The number of records within the page.

- `DATA_SIZE`

  The sum of the sizes of the records. This column is applicable only to pages with a `PAGE_TYPE` value of `INDEX`.

- `COMPRESSED_SIZE`

  The compressed page size. `NULL` for pages that are not compressed.

- `COMPRESSED`

  Whether the page is compressed.

- `IO_FIX`

  Whether any I/O is pending for this page: `IO_NONE` = no pending I/O, `IO_READ` = read pending, `IO_WRITE` = write pending.

- `IS_OLD`

  Whether the block is in the sublist of old blocks in the LRU list.

- `FREE_PAGE_CLOCK`

  The value of the `freed_page_clock` counter when the block was the last placed at the head of the LRU list. The `freed_page_clock` counter tracks the number of blocks removed from the end of the LRU list.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_PAGE_LRU LIMIT 1\G
*************************** 1. row ***************************
           POOL_ID: 0
```

```
       LRU_POSITION: 0
              SPACE: 97
        PAGE_NUMBER: 1984
          PAGE_TYPE: INDEX
         FLUSH_TYPE: 1
          FIX_COUNT: 0
          IS_HASHED: YES
NEWEST_MODIFICATION: 719490396
OLDEST_MODIFICATION: 0
        ACCESS_TIME: 3378383796
         TABLE_NAME: `employees`.`salaries`
         INDEX_NAME: PRIMARY
     NUMBER_RECORDS: 468
          DATA_SIZE: 14976
    COMPRESSED_SIZE: 0
         COMPRESSED: NO
             IO_FIX: IO_NONE
             IS_OLD: YES
    FREE_PAGE_CLOCK: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- Querying this table can require MySQL to allocate a large block of contiguous memory, more than 64 bytes times the number of active pages in the buffer pool. This allocation could potentially cause an out-of-memory error, especially for systems with multi-gigabyte buffer pools.
- Querying this table requires MySQL to lock the data structure representing the buffer pool while traversing the LRU list, which can reduce concurrency, especially for systems with multi-gigabyte buffer pools.
- When tables, table rows, partitions, or indexes are deleted, associated pages remain in the buffer pool until space is required for other data. The `INNODB_BUFFER_PAGE_LRU` table reports information about these pages until they are evicted from the buffer pool. For more information about how the `InnoDB` manages buffer pool data, see Buffer Pool.

5.4 The INFORMATION_SCHEMA INNODB_BUFFER_POOL_STATS Table

The `INNODB_BUFFER_POOL_STATS` table provides much of the same buffer pool information provided in `SHOW ENGINE INNODB STATUS` output. Much of the same information may also be obtained using `InnoDB` buffer pool server status variables.

The idea of making pages in the buffer pool “young” or “not young” refers to transferring them between the sublists at the head and tail of the buffer pool data structure. Pages made “young” take longer to age out of the buffer pool, while pages made “not young” are moved much closer to the point of eviction.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Buffer Pool Tables.

The `INNODB_BUFFER_POOL_STATS` table has these columns:

- `POOL_ID`

The buffer pool ID. This is an identifier to distinguish between multiple buffer pool instances.

- `POOL_SIZE`

  The `InnoDB` buffer pool size in pages.

- `FREE_BUFFERS`

  The number of free pages in the `InnoDB` buffer pool.

- `DATABASE_PAGES`

  The number of pages in the `InnoDB` buffer pool containing data. This number includes both dirty and clean pages.

- `OLD_DATABASE_PAGES`

  The number of pages in the `old` buffer pool sublist.

- `MODIFIED_DATABASE_PAGES`

  The number of modified (dirty) database pages.

- `PENDING_DECOMPRESS`

  The number of pages pending decompression.

- `PENDING_READS`

  The number of pending reads.

- `PENDING_FLUSH_LRU`

  The number of pages pending flush in the LRU.

- `PENDING_FLUSH_LIST`

  The number of pages pending flush in the flush list.

- `PAGES_MADE_YOUNG`

  The number of pages made young.

- `PAGES_NOT_MADE_YOUNG`

  The number of pages not made young.

- `PAGES_MADE_YOUNG_RATE`

  The number of pages made young per second (pages made young since the last printout / time elapsed).

- `PAGES_MADE_NOT_YOUNG_RATE`

  The number of pages not made per second (pages not made young since the last printout / time elapsed).

- `NUMBER_PAGES_READ`

The number of pages read.

- `NUMBER_PAGES_CREATED`

  The number of pages created.

- `NUMBER_PAGES_WRITTEN`

  The number of pages written.

- `PAGES_READ_RATE`

  The number of pages read per second (pages read since the last printout / time elapsed).

- `PAGES_CREATE_RATE`

  The number of pages created per second (pages created since the last printout / time elapsed).

- `PAGES_WRITTEN_RATE`

  The number of pages written per second (pages written since the last printout / time elapsed).

- `NUMBER_PAGES_GET`

  The number of logical read requests.

- `HIT_RATE`

  The buffer pool hit rate.

- `YOUNG_MAKE_PER_THOUSAND_GETS`

  The number of pages made young per thousand gets.

- `NOT_YOUNG_MAKE_PER_THOUSAND_GETS`

  The number of pages not made young per thousand gets.

- `NUMBER_PAGES_READ_AHEAD`

  The number of pages read ahead.

- `NUMBER_READ_AHEAD_EVICTED`

  The number of pages read into the `InnoDB` buffer pool by the read-ahead background thread that were subsequently evicted without having been accessed by queries.

- `READ_AHEAD_RATE`

  The read-ahead rate per second (pages read ahead since the last printout / time elapsed).

- `READ_AHEAD_EVICTED_RATE`

  The number of read-ahead pages evicted without access per second (read-ahead pages not accessed since the last printout / time elapsed).

- `LRU_IO_TOTAL`

  Total LRU I/O.

- `LRU_IO_CURRENT`

  LRU I/O for the current interval.

- `UNCOMPRESS_TOTAL`

  The total number of pages decompressed.

- `UNCOMPRESS_CURRENT`

  The number of pages decompressed in the current interval.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_BUFFER_POOL_STATS\G
*************************** 1. row ***************************
                         POOL_ID: 0
                       POOL_SIZE: 8192
                    FREE_BUFFERS: 1
                  DATABASE_PAGES: 8085
              OLD_DATABASE_PAGES: 2964
         MODIFIED_DATABASE_PAGES: 0
              PENDING_DECOMPRESS: 0
                   PENDING_READS: 0
               PENDING_FLUSH_LRU: 0
              PENDING_FLUSH_LIST: 0
                PAGES_MADE_YOUNG: 22821
            PAGES_NOT_MADE_YOUNG: 3544303
           PAGES_MADE_YOUNG_RATE: 357.62602199870594
       PAGES_MADE_NOT_YOUNG_RATE: 0
               NUMBER_PAGES_READ: 2389
            NUMBER_PAGES_CREATED: 12385
            NUMBER_PAGES_WRITTEN: 13111
                 PAGES_READ_RATE: 0
              PAGES_CREATE_RATE: 0
              PAGES_WRITTEN_RATE: 0
                NUMBER_PAGES_GET: 33322210
                        HIT_RATE: 1000
    YOUNG_MAKE_PER_THOUSAND_GETS: 18
NOT_YOUNG_MAKE_PER_THOUSAND_GETS: 0
         NUMBER_PAGES_READ_AHEAD: 2024
       NUMBER_READ_AHEAD_EVICTED: 0
                 READ_AHEAD_RATE: 0
         READ_AHEAD_EVICTED_RATE: 0
                    LRU_IO_TOTAL: 0
                  LRU_IO_CURRENT: 0
                UNCOMPRESS_TOTAL: 0
              UNCOMPRESS_CURRENT: 0
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.5 The INFORMATION_SCHEMA INNODB_CACHED_INDEXES Table

The `INNODB_CACHED_INDEXES` table reports the number of index pages cached in the `InnoDB` buffer pool for each index.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Buffer Pool Tables.

The INNODB_CACHED_INDEXES table has these columns:

- SPACE_ID

  The tablespace ID.

- INDEX_ID

  An identifier for the index. Index identifiers are unique across all the databases in an instance.

- N_CACHED_PAGES

  The number of index pages cached in the InnoDB buffer pool.

Examples

This query returns the number of index pages cached in the InnoDB buffer pool for a specific index:

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CACHED_INDEXES WHERE INDEX_ID=65\G
*************************** 1. row ***************************
      SPACE_ID: 4294967294
      INDEX_ID: 65
N_CACHED_PAGES: 45
```

This query returns the number of index pages cached in the InnoDB buffer pool for each index, using the INNODB_INDEXES and INNODB_TABLES tables to resolve the table name and index name for each INDEX_ID value.

```
SELECT
  tables.NAME AS table_name,
  indexes.NAME AS index_name,
  cached.N_CACHED_PAGES AS n_cached_pages
FROM
  INFORMATION_SCHEMA.INNODB_CACHED_INDEXES AS cached,
  INFORMATION_SCHEMA.INNODB_INDEXES AS indexes,
  INFORMATION_SCHEMA.INNODB_TABLES AS tables
WHERE
  cached.INDEX_ID = indexes.INDEX_ID
  AND indexes.TABLE_ID = tables.TABLE_ID;
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

5.6 The INFORMATION_SCHEMA INNODB_CMP and INNODB_CMP_RESET Tables

The INNODB_CMP and INNODB_CMP_RESET tables provide status information on operations related to compressed InnoDB tables.

The INNODB_CMP and INNODB_CMP_RESET tables have these columns:

- PAGE_SIZE

The compressed page size in bytes.

- `COMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been compressed. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- `COMPRESS_OPS_OK`

  The number of times a B-tree page of size `PAGE_SIZE` has been successfully compressed. This count should never exceed `COMPRESS_OPS`.

- `COMPRESS_TIME`

  The total time in seconds used for attempts to compress B-tree pages of size `PAGE_SIZE`.

- `UNCOMPRESS_OPS`

  The number of times a B-tree page of size `PAGE_SIZE` has been uncompressed. B-tree pages are uncompressed whenever compression fails or at first access when the uncompressed page does not exist in the buffer pool.

- `UNCOMPRESS_TIME`

  The total time in seconds used for uncompressing B-tree pages of the size `PAGE_SIZE`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP\G
*************************** 1. row ***************************
      page_size: 1024
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 2. row ***************************
      page_size: 2048
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 3. row ***************************
      page_size: 4096
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 0
uncompress_time: 0
*************************** 4. row ***************************
      page_size: 8192
   compress_ops: 86955
compress_ops_ok: 81182
  compress_time: 27
 uncompress_ops: 26828
uncompress_time: 5
*************************** 5. row ***************************
      page_size: 16384
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
```

```
 uncompress_ops: 0
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Monitoring InnoDB Table Compression at Runtime and Using the Compression Information Schema Tables. For general information about `InnoDB` table compression, see InnoDB Table and Page Compression.

5.7 The INFORMATION_SCHEMA INNODB_CMPMEM and INNODB_CMPMEM_RESET Tables

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables provide status information on compressed pages within the `InnoDB` buffer pool.

The `INNODB_CMPMEM` and `INNODB_CMPMEM_RESET` tables have these columns:

- `PAGE_SIZE`

  The block size in bytes. Each record of this table describes blocks of this size.

- `BUFFER_POOL_INSTANCE`

  A unique identifier for the buffer pool instance.

- `PAGES_USED`

  The number of blocks of size `PAGE_SIZE` that are currently in use.

- `PAGES_FREE`

  The number of blocks of size `PAGE_SIZE` that are currently available for allocation. This column shows the external fragmentation in the memory pool. Ideally, these numbers should be at most 1.

- `RELOCATION_OPS`

  The number of times a block of size `PAGE_SIZE` has been relocated. The buddy system can relocate the allocated “buddy neighbor” of a freed block when it tries to form a bigger freed block. Reading from the `INNODB_CMPMEM_RESET` table resets this count.

- `RELOCATION_TIME`

  The total time in microseconds used for relocating blocks of size `PAGE_SIZE`. Reading from the table `INNODB_CMPMEM_RESET` resets this count.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMPMEM\G
*************************** 1. row ***************************
          page_size: 1024
buffer_pool_instance: 0
         pages_used: 0
```

```
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 2. row ***************************
           page_size: 2048
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 3. row ***************************
           page_size: 4096
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
*************************** 4. row ***************************
           page_size: 8192
buffer_pool_instance: 0
          pages_used: 7673
          pages_free: 15
      relocation_ops: 4638
     relocation_time: 0
*************************** 5. row ***************************
           page_size: 16384
buffer_pool_instance: 0
          pages_used: 0
          pages_free: 0
      relocation_ops: 0
     relocation_time: 0
```

Notes

- Use these tables to measure the effectiveness of `InnoDB` table compression in your database.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For usage information, see Monitoring InnoDB Table Compression at Runtime and Using the Compression Information Schema Tables. For general information about `InnoDB` table compression, see InnoDB Table and Page Compression.

5.8 The INFORMATION_SCHEMA INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET Tables

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables provide status information on operations related to compressed `InnoDB` tables and indexes, with separate statistics for each combination of database, table, and index, to help you evaluate the performance and usefulness of compression for specific tables.

For a compressed `InnoDB` table, both the table data and all the secondary indexes are compressed. In this context, the table data is treated as just another index, one that happens to contain all the columns: the clustered index.

The `INNODB_CMP_PER_INDEX` and `INNODB_CMP_PER_INDEX_RESET` tables have these columns:

- `DATABASE_NAME`

  The schema (database) containing the applicable table.

- TABLE_NAME

  The table to monitor for compression statistics.

- INDEX_NAME

  The index to monitor for compression statistics.

- COMPRESS_OPS

  The number of compression operations attempted. Pages are compressed whenever an empty page is created or the space for the uncompressed modification log runs out.

- COMPRESS_OPS_OK

  The number of successful compression operations. Subtract from the COMPRESS_OPS value to get the number of compression failures. Divide by the COMPRESS_OPS value to get the percentage of compression failures.

- COMPRESS_TIME

  The total time in seconds used for compressing data in this index.

- UNCOMPRESS_OPS

  The number of uncompression operations performed. Compressed InnoDB pages are uncompressed whenever compression fails, or the first time a compressed page is accessed in the buffer pool and the uncompressed page does not exist.

- UNCOMPRESS_TIME

  The total time in seconds used for uncompressing data in this index.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_CMP_PER_INDEX\G
*************************** 1. row ***************************
  database_name: employees
     table_name: salaries
     index_name: PRIMARY
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 23451
uncompress_time: 4
*************************** 2. row ***************************
  database_name: employees
     table_name: salaries
     index_name: emp_no
   compress_ops: 0
compress_ops_ok: 0
  compress_time: 0
 uncompress_ops: 1597
uncompress_time: 0
```

Notes

- Use these tables to measure the effectiveness of InnoDB table compression for specific tables, indexes, or both.
- You must have the PROCESS privilege to query these tables.

- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of these tables, including data types and default values.
- Because collecting separate measurements for every index imposes substantial performance overhead, INNODB_CMP_PER_INDEX and INNODB_CMP_PER_INDEX_RESET statistics are not gathered by default. You must enable the innodb_cmp_per_index_enabled system variable before performing the operations on compressed tables that you want to monitor.
- For usage information, see Monitoring InnoDB Table Compression at Runtime and Using the Compression Information Schema Tables. For general information about InnoDB table compression, see InnoDB Table and Page Compression.

5.9 The INFORMATION_SCHEMA INNODB_COLUMNS Table

The INNODB_COLUMNS table provides metadata about InnoDB table columns.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The INNODB_COLUMNS table has these columns:

- TABLE_ID

  An identifier representing the table associated with the column; the same value as INNODB_TABLES.TABLE_ID.

- NAME

  The name of the column. These names can be uppercase or lowercase depending on the lower_case_table_names setting. There are no special system-reserved names for columns.

- POS

  The ordinal position of the column within the table, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps. The POS value for a virtual generated column encodes the column sequence number and ordinal position of the column. For more information, see the POS column description in Section 5.29, “The INFORMATION_SCHEMA INNODB_VIRTUAL Table”.

- MTYPE

  Stands for “main type”. A numeric identifier for the column type. 1 = VARCHAR, 2 = CHAR, 3 = FIXBINARY, 4 = BINARY, 5 = BLOB, 6 = INT, 7 = SYS_CHILD, 8 = SYS, 9 = FLOAT, 10 = DOUBLE, 11 = DECIMAL, 12 = VARMYSQL, 13 = MYSQL, 14 = GEOMETRY.

- PRTYPE

  The InnoDB “precise type”, a binary value with bits representing MySQL data type, character set code, and nullability.

- LEN

  The column length, for example 4 for INT and 8 for BIGINT. For character columns in multibyte character sets, this length value is the maximum length in bytes needed to represent a definition such as VARCHAR(*N*); that is, it might be 2*N, 3*N, and so on depending on the character encoding.

- HAS_DEFAULT

A boolean value indicating whether a column that was added instantly using `ALTER TABLE ... ADD COLUMN` with `ALGORITHM=INSTANT` has a default value. All columns added instantly have a default value, which makes this column an indicator of whether the column was added instantly.

- `DEFAULT_VALUE`

  The initial default value of a column that was added instantly using `ALTER TABLE ... ADD COLUMN` with `ALGORITHM=INSTANT`. If the default value is `NULL` or was not specified, this column reports `NULL`. An explicitly specified non-`NULL` default value is shown in an internal binary format. Subsequent modifications of the column default value do not change the value reported by this column.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_COLUMNS where TABLE_ID = 71\G
*************************** 1. row ***************************
     TABLE_ID: 71
         NAME: col1
          POS: 0
        MTYPE: 6
       PRTYPE: 1027
          LEN: 4
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 2. row ***************************
     TABLE_ID: 71
         NAME: col2
          POS: 1
        MTYPE: 2
       PRTYPE: 524542
          LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
*************************** 3. row ***************************
     TABLE_ID: 71
         NAME: col3
          POS: 2
        MTYPE: 1
       PRTYPE: 524303
          LEN: 10
  HAS_DEFAULT: 0
DEFAULT_VALUE: NULL
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.10 The INFORMATION_SCHEMA INNODB_DATAFILES Table

The `INNODB_DATAFILES` table provides data file path information for `InnoDB` file-per-table and general tablespaces.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

> **Note**
>
> The INFORMATION_SCHEMA FILES table reports metadata for InnoDB tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, the global temporary tablespace, and undo tablespaces.

The INNODB_DATAFILES table has these columns:

- SPACE

  The tablespace ID.

- PATH

  The tablespace data file path. If a file-per-table tablespace is created in a location outside the MySQL data directory, the path value is a fully qualified directory path. Otherwise, the path is relative to the data directory.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_DATAFILES WHERE SPACE = 57\G
*************************** 1. row ***************************
SPACE: 57
 PATH: ./test/t1.ibd
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

5.11 The INFORMATION_SCHEMA INNODB_FIELDS Table

The INNODB_FIELDS table provides metadata about the key columns (fields) of InnoDB indexes.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The INNODB_FIELDS table has these columns:

- INDEX_ID

  An identifier for the index associated with this key field; the same value as INNODB_INDEXES.INDEX_ID.

- NAME

  The name of the original column from the table; the same value as INNODB_COLUMNS.NAME.

- POS

  The ordinal position of the key field within the index, starting from 0 and incrementing sequentially. When a column is dropped, the remaining columns are reordered so that the sequence has no gaps.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FIELDS WHERE INDEX_ID = 117\G
*************************** 1. row ***************************
```

```
INDEX_ID: 117
    NAME: col1
     POS: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.12 The INFORMATION_SCHEMA INNODB_FOREIGN Table

The `INNODB_FOREIGN` table provides metadata about `InnoDB` foreign keys.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The `INNODB_FOREIGN` table has these columns:

- `ID`

  The name (not a numeric value) of the foreign key index, preceded by the schema (database) name (for example, `test/products_fk`).

- `FOR_NAME`

  The name of the child table in this foreign key relationship.

- `REF_NAME`

  The name of the parent table in this foreign key relationship.

- `N_COLS`

  The number of columns in the foreign key index.

- `TYPE`

  A collection of bit flags with information about the foreign key column, ORed together. 0 = `ON DELETE/UPDATE RESTRICT`, 1 = `ON DELETE CASCADE`, 2 = `ON DELETE SET NULL`, 4 = `ON UPDATE CASCADE`, 8 = `ON UPDATE SET NULL`, 16 = `ON DELETE NO ACTION`, 32 = `ON UPDATE NO ACTION`.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN\G
*************************** 1. row ***************************
      ID: test/fk1
FOR_NAME: test/child
REF_NAME: test/parent
  N_COLS: 1
    TYPE: 1
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.13 The INFORMATION_SCHEMA INNODB_FOREIGN_COLS Table

The `INNODB_FOREIGN_COLS` table provides status information about `InnoDB` foreign key columns.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The `INNODB_FOREIGN_COLS` table has these columns:

- `ID`

  The foreign key index associated with this index key field; the same value as `INNODB_FOREIGN.ID`.

- `FOR_COL_NAME`

  The name of the associated column in the child table.

- `REF_COL_NAME`

  The name of the associated column in the parent table.

- `POS`

  The ordinal position of this key field within the foreign key index, starting from 0.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FOREIGN_COLS WHERE ID = 'test/fk1'\G
*************************** 1. row ***************************
          ID: test/fk1
FOR_COL_NAME: parent_id
REF_COL_NAME: id
         POS: 0
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.14 The INFORMATION_SCHEMA INNODB_FT_BEING_DELETED Table

The `INNODB_FT_BEING_DELETED` table provides a snapshot of the `INNODB_FT_DELETED` table; it is used only during an `OPTIMIZE TABLE` maintenance operation. When `OPTIMIZE TABLE` is run, the `INNODB_FT_BEING_DELETED` table is emptied, and `DOC_ID` values are removed from the `INNODB_FT_DELETED` table. Because the contents of `INNODB_FT_BEING_DELETED` typically have a short lifetime, this table has limited utility for monitoring or debugging. For information about running `OPTIMIZE TABLE` on tables with `FULLTEXT` indexes, see Fine-Tuning MySQL Full-Text Search.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`). The output appears similar to the example provided for the `INNODB_FT_DELETED` table.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The INNODB_FT_BEING_DELETED table has these columns:

- DOC_ID

  The document ID of the row that is in the process of being deleted. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table contains no suitable column. This value is used when you perform text searches, to skip rows in the INNODB_FT_INDEX_TABLE table before data for deleted rows is physically removed from the FULLTEXT index by an OPTIMIZE TABLE statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Notes

- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- You must have the PROCESS privilege to query this table.
- For more information about InnoDB FULLTEXT search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.15 The INFORMATION_SCHEMA INNODB_FT_CONFIG Table

The INNODB_FT_CONFIG table provides metadata about the FULLTEXT index and associated processing for an InnoDB table.

This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example, test/articles).

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The INNODB_FT_CONFIG table has these columns:

- KEY

  The name designating an item of metadata for an InnoDB table containing a FULLTEXT index.

  The values for this column might change, depending on the needs for performance tuning and debugging for InnoDB full-text processing. The key names and their meanings include:

  - optimize_checkpoint_limit: The number of seconds after which an OPTIMIZE TABLE run stops.
  - synced_doc_id: The next DOC_ID to be issued.
  - stopword_table_name: The *database/table* name for a user-defined stopword table. The VALUE column is empty if there is no user-defined stopword table.
  - use_stopword: Indicates whether a stopword table is used, which is defined when the FULLTEXT index is created.

- VALUE

The value associated with the corresponding KEY column, reflecting some limit or current value for an aspect of a FULLTEXT index for an InnoDB table.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_CONFIG;
+---------------------------+-------------------+
| KEY                       | VALUE             |
+---------------------------+-------------------+
| optimize_checkpoint_limit | 180               |
| synced_doc_id             | 0                 |
| stopword_table_name       | test/my_stopwords |
| use_stopword              | 1                 |
+---------------------------+-------------------+
```

Notes

- This table is intended only for internal configuration. It is not intended for statistical information purposes.
- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- For more information about InnoDB FULLTEXT search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.16 The INFORMATION_SCHEMA INNODB_FT_DEFAULT_STOPWORD Table

The INNODB_FT_DEFAULT_STOPWORD table holds a list of stopwords that are used by default when creating a FULLTEXT index on InnoDB tables. For information about the default InnoDB stopword list and how to define your own stopword lists, see Full-Text Stopwords.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The INNODB_FT_DEFAULT_STOPWORD table has these columns:

- value

  A word that is used by default as a stopword for FULLTEXT indexes on InnoDB tables. This is not used if you override the default stopword processing with either the innodb_ft_server_stopword_table or the innodb_ft_user_stopword_table system variable.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DEFAULT_STOPWORD;
+-------+
| value |
+-------+
| a     |
| about |
| an    |
| are   |
| as    |
| at    |
```

```
| be    |
| by    |
| com   |
| de    |
| en    |
| for   |
| from  |
| how   |
| i     |
| in    |
| is    |
| it    |
| la    |
| of    |
| on    |
| or    |
| that  |
| the   |
| this  |
| to    |
| was   |
| what  |
| when  |
| where |
| who   |
| will  |
| with  |
| und   |
| the   |
| www   |
+-------+
36 rows in set (0.00 sec)
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.17 The INFORMATION_SCHEMA INNODB_FT_DELETED Table

The `INNODB_FT_DELETED` table stores rows that are deleted from the `FULLTEXT` index for an `InnoDB` table. To avoid expensive index reorganization during DML operations for an `InnoDB FULLTEXT` index, the information about newly deleted words is stored separately, filtered out of search results when you do a text search, and removed from the main search index only when you issue an `OPTIMIZE TABLE` statement for the `InnoDB` table. For more information, see Optimizing InnoDB Full-Text Indexes.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The `INNODB_FT_DELETED` table has these columns:

- `DOC_ID`

The document ID of the newly deleted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column. This value is used when you perform text searches, to skip rows in the `INNODB_FT_INDEX_TABLE` table before data for deleted rows is physically removed from the `FULLTEXT` index by an `OPTIMIZE TABLE` statement. For more information, see Optimizing InnoDB Full-Text Indexes.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_FT_DELETED;
+--------+
| DOC_ID |
+--------+
|      6 |
|      7 |
|      8 |
+--------+
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.18 The INFORMATION_SCHEMA INNODB_FT_INDEX_CACHE Table

The `INNODB_FT_INDEX_CACHE` table provides token information about newly inserted rows in a `FULLTEXT` index. To avoid expensive index reorganization during DML operations, the information about newly indexed words is stored separately, and combined with the main search index only when `OPTIMIZE TABLE` is run, when the server is shut down, or when the cache size exceeds a limit defined by the `innodb_ft_cache_size` or `innodb_ft_total_cache_size` system variable.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The `INNODB_FT_INDEX_CACHE` table has these columns:

- `WORD`

  A word extracted from the text of a newly inserted row.

- `FIRST_DOC_ID`

  The first document ID in which this word appears in the `FULLTEXT` index.

- `LAST_DOC_ID`

The last document ID in which this word appears in the FULLTEXT index.

- DOC_COUNT

  The number of rows in which this word appears in the FULLTEXT index. The same word can occur several times within the cache table, once for each combination of DOC_ID and POSITION values.

- DOC_ID

  The document ID of the newly inserted row. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by InnoDB when the table contains no suitable column.

- POSITION

  The position of this particular instance of the word within the relevant document identified by the DOC_ID value. The value does not represent an absolute position; it is an offset added to the POSITION of the previous instance of that word.

Notes

- This table is empty initially. Before querying it, set the value of the innodb_ft_aux_table system variable to the name (including the database name) of the table that contains the FULLTEXT index (for example test/articles). The following example demonstrates how to use the innodb_ft_aux_table system variable to show information about a FULLTEXT index for a specified table.

```
mysql> USE test;
mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
         FULLTEXT (title,body)
       ) ENGINE=InnoDB;
mysql> INSERT INTO articles (title,body) VALUES
       ('MySQL Tutorial','DBMS stands for DataBase ...'),
       ('How To Use MySQL Well','After you went through a ...'),
       ('Optimizing MySQL','In this tutorial we show ...'),
       ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
       ('MySQL vs. YourSQL','In the following database comparison ...'),
       ('MySQL Security','When configured properly, MySQL ...');
mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';
mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
       FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_CACHE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+
```

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.
- For more information about InnoDB FULLTEXT search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.19 The INFORMATION_SCHEMA INNODB_FT_INDEX_TABLE Table

The `INNODB_FT_INDEX_TABLE` table provides information about the inverted index used to process text searches against the `FULLTEXT` index of an `InnoDB` table.

This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`).

For related usage information and examples, see InnoDB INFORMATION_SCHEMA FULLTEXT Index Tables.

The `INNODB_FT_INDEX_TABLE` table has these columns:

- `WORD`

  A word extracted from the text of the columns that are part of a `FULLTEXT`.

- `FIRST_DOC_ID`

  The first document ID in which this word appears in the `FULLTEXT` index.

- `LAST_DOC_ID`

  The last document ID in which this word appears in the `FULLTEXT` index.

- `DOC_COUNT`

  The number of rows in which this word appears in the `FULLTEXT` index. The same word can occur several times within the cache table, once for each combination of `DOC_ID` and `POSITION` values.

- `DOC_ID`

  The document ID of the row containing the word. This value might reflect the value of an ID column that you defined for the underlying table, or it can be a sequence value generated by `InnoDB` when the table contains no suitable column.

- `POSITION`

  The position of this particular instance of the word within the relevant document identified by the `DOC_ID` value.

Notes

- This table is empty initially. Before querying it, set the value of the `innodb_ft_aux_table` system variable to the name (including the database name) of the table that contains the `FULLTEXT` index (for example, `test/articles`). The following example demonstrates how to use the `innodb_ft_aux_table` system variable to show information about a `FULLTEXT` index for a specified table. Before information for newly inserted rows appears in `INNODB_FT_INDEX_TABLE`, the `FULLTEXT` index cache must be flushed to disk. This is accomplished by running an `OPTIMIZE TABLE` operation on the indexed table with the `innodb_optimize_fulltext_only` system variable enabled. (The example disables that variable again at the end because it is intended to be enabled only temporarily.)

```
mysql> USE test;
mysql> CREATE TABLE articles (
         id INT UNSIGNED AUTO_INCREMENT NOT NULL PRIMARY KEY,
         title VARCHAR(200),
         body TEXT,
```

```
        FULLTEXT (title,body)
      ) ENGINE=InnoDB;
mysql> INSERT INTO articles (title,body) VALUES
      ('MySQL Tutorial','DBMS stands for DataBase ...'),
      ('How To Use MySQL Well','After you went through a ...'),
      ('Optimizing MySQL','In this tutorial we show ...'),
      ('1001 MySQL Tricks','1. Never run mysqld as root. 2. ...'),
      ('MySQL vs. YourSQL','In the following database comparison ...'),
      ('MySQL Security','When configured properly, MySQL ...');
mysql> SET GLOBAL innodb_optimize_fulltext_only=ON;
mysql> OPTIMIZE TABLE articles;
+--------------+----------+----------+----------+
| Table        | Op       | Msg_type | Msg_text |
+--------------+----------+----------+----------+
| test.articles | optimize | status   | OK       |
+--------------+----------+----------+----------+
mysql> SET GLOBAL innodb_ft_aux_table = 'test/articles';
mysql> SELECT WORD, DOC_COUNT, DOC_ID, POSITION
      FROM INFORMATION_SCHEMA.INNODB_FT_INDEX_TABLE LIMIT 5;
+------------+-----------+--------+----------+
| WORD       | DOC_COUNT | DOC_ID | POSITION |
+------------+-----------+--------+----------+
| 1001       |         1 |      4 |        0 |
| after      |         1 |      2 |       22 |
| comparison |         1 |      5 |       44 |
| configured |         1 |      6 |       20 |
| database   |         2 |      1 |       31 |
+------------+-----------+--------+----------+
mysql> SET GLOBAL innodb_optimize_fulltext_only=OFF;
```

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.
- For more information about `InnoDB FULLTEXT` search, see InnoDB Full-Text Indexes, and Full-Text Search Functions.

5.20 The INFORMATION_SCHEMA INNODB_INDEXES Table

The `INNODB_INDEXES` table provides metadata about `InnoDB` indexes.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The `INNODB_INDEXES` table has these columns:

- `INDEX_ID`

  An identifier for the index. Index identifiers are unique across all the databases in an instance.

- `NAME`

  The name of the index. Most indexes created implicitly by `InnoDB` have consistent names but the index names are not necessarily unique. Examples: `PRIMARY` for a primary key index, `GEN_CLUST_INDEX` for the index representing a primary key when one is not specified, and `ID_IND`, `FOR_IND`, and `REF_IND` for foreign key constraints.

- `TABLE_ID`

  An identifier representing the table associated with the index; the same value as `INNODB_TABLES.TABLE_ID`.

- `TYPE`

  A numeric value derived from bit-level information that identifies the index type. 0 = nonunique secondary index; 1 = automatically generated clustered index (`GEN_CLUST_INDEX`); 2 = unique nonclustered index; 3 = clustered index; 32 = full-text index; 64 = spatial index; 128 = secondary index on a virtual generated column.

- `N_FIELDS`

  The number of columns in the index key. For `GEN_CLUST_INDEX` indexes, this value is 0 because the index is created using an artificial value rather than a real table column.

- `PAGE_NO`

  The root page number of the index B-tree. For full-text indexes, the `PAGE_NO` column is unused and set to -1 (`FIL_NULL`) because the full-text index is laid out in several B-trees (auxiliary tables).

- `SPACE`

  An identifier for the tablespace where the index resides. 0 means the `InnoDB` system tablespace. Any other number represents a table created with a separate `.ibd` file in file-per-table mode. This identifier stays the same after a `TRUNCATE TABLE` statement. Because all indexes for a table reside in the same tablespace as the table, this value is not necessarily unique.

- `MERGE_THRESHOLD`

  The merge threshold value for index pages. If the amount of data in an index page falls below the `MERGE_THRESHOLD` value when a row is deleted or when a row is shortened by an update operation, `InnoDB` attempts to merge the index page with the neighboring index page. The default threshold value is 50%. For more information, see Configuring the Merge Threshold for Index Pages.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_INDEXES WHERE TABLE_ID = 34\G
*************************** 1. row ***************************
       INDEX_ID: 39
           NAME: GEN_CLUST_INDEX
       TABLE_ID: 34
           TYPE: 1
       N_FIELDS: 0
        PAGE_NO: 3
          SPACE: 23
MERGE_THRESHOLD: 50
*************************** 2. row ***************************
       INDEX_ID: 40
           NAME: i1
       TABLE_ID: 34
           TYPE: 0
       N_FIELDS: 1
        PAGE_NO: 4
          SPACE: 23
MERGE_THRESHOLD: 50
```

Notes

- You must have the `PROCESS` privilege to query this table.

- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.21 The INFORMATION_SCHEMA INNODB_METRICS Table

The `INNODB_METRICS` table provides a wide variety of `InnoDB` performance information, complementing the specific focus areas of the Performance Schema tables for `InnoDB`. With simple queries, you can check the overall health of the system. With more detailed queries, you can diagnose issues such as performance bottlenecks, resource shortages, and application issues.

Each monitor represents a point within the `InnoDB` source code that is instrumented to gather counter information. Each counter can be started, stopped, and reset. You can also perform these actions for a group of counters using their common module name.

By default, relatively little data is collected. To start, stop, and reset counters, set one of the system variables `innodb_monitor_enable`, `innodb_monitor_disable`, `innodb_monitor_reset`, or `innodb_monitor_reset_all`, using the name of the counter, the name of the module, a wildcard match for such a name using the "%" character, or the special keyword `all`.

For usage information, see InnoDB INFORMATION_SCHEMA Metrics Table.

The `INNODB_METRICS` table has these columns:

- `NAME`

  A unique name for the counter.

- `SUBSYSTEM`

  The aspect of `InnoDB` that the metric applies to.

- `COUNT`

  The value since the counter was enabled.

- `MAX_COUNT`

  The maximum value since the counter was enabled.

- `MIN_COUNT`

  The minimum value since the counter was enabled.

- `AVG_COUNT`

  The average value since the counter was enabled.

- `COUNT_RESET`

  The counter value since it was last reset. (The `_RESET` columns act like the lap counter on a stopwatch: you can measure the activity during some time interval, while the cumulative figures are still available in `COUNT`, `MAX_COUNT`, and so on.)

- `MAX_COUNT_RESET`

  The maximum counter value since it was last reset.

- `MIN_COUNT_RESET`

  The minimum counter value since it was last reset.

- `AVG_COUNT_RESET`

  The average counter value since it was last reset.

- `TIME_ENABLED`

  The timestamp of the last start.

- `TIME_DISABLED`

  The timestamp of the last stop.

- `TIME_ELAPSED`

  The elapsed time in seconds since the counter started.

- `TIME_RESET`

  The timestamp of the last reset.

- `STATUS`

  Whether the counter is still running (`enabled`) or stopped (`disabled`).

- `TYPE`

  Whether the item is a cumulative counter, or measures the current value of some resource.

- `COMMENT`

  The counter description.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_METRICS WHERE NAME='dml_inserts'\G
*************************** 1. row ***************************
           NAME: dml_inserts
      SUBSYSTEM: dml
          COUNT: 3
      MAX_COUNT: 3
      MIN_COUNT: NULL
      AVG_COUNT: 0.046153846153846156
    COUNT_RESET: 3
MAX_COUNT_RESET: 3
MIN_COUNT_RESET: NULL
AVG_COUNT_RESET: NULL
   TIME_ENABLED: 2014-12-04 14:18:28
  TIME_DISABLED: NULL
   TIME_ELAPSED: 65
     TIME_RESET: NULL
         STATUS: enabled
           TYPE: status_counter
        COMMENT: Number of rows inserted
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

- Transaction counter `COUNT` values may differ from the number of transaction events reported in Performance Schema `EVENTS_TRANSACTIONS_SUMMARY` tables. `InnoDB` counts only those transactions that it executes, whereas Performance Schema collects events for all non-aborted transactions initiated by the server, including empty transactions.

5.22 The INFORMATION_SCHEMA INNODB_SESSION_TEMP_TABLESPACES Table

The `INNODB_SESSION_TEMP_TABLESPACES` table provides metadata about session temporary tablespaces used for internal and user-created temporary tables. This table was added in MySQL 8.0.13.

The `INNODB_SESSION_TEMP_TABLESPACES` table has these columns:

- `ID`

  The process or session ID.

- `SPACE`

  The tablespace ID. A range of 400 thousand space IDs is reserved for session temporary tablespaces. Session temporary tablespaces are recreated each time the server is started. Space IDs are not persisted when the server is shut down and may be reused.

- `PATH`

  The tablespace data file path. A session temporary tablespace has an `ibt` file extension.

- `SIZE`

  The size of the tablespace, in bytes.

- `STATE`

  The state of the tablespace. `ACTIVE` indicates that the tablespace is currently used by a session. `INACTIVE` indicates that the tablespace is in the pool of available session temporary tablespaces.

- `PURPOSE`

  The purpose of the tablespace. `INTRINSIC` indicates that the tablespace is used for optimized internal temporary tables use by the optimizer. `SLAVE` indicates that the tablespace is allocated for storing user-created temporary tables on a replication slave. `USER` indicates that the tablespace is used for user-created temporary tables. `NONE` indicates that the tablespace is not in use.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_SESSION_TEMP_TABLESPACES;
+----+-----------+---------------------------+-------+----------+-----------+
| ID | SPACE     | PATH                      | SIZE  | STATE    | PURPOSE   |
+----+-----------+---------------------------+-------+----------+-----------+
|  8 | 4294566162 | ./#innodb_temp/temp_10.ibt | 81920 | ACTIVE   | INTRINSIC |
|  8 | 4294566161 | ./#innodb_temp/temp_9.ibt  | 98304 | ACTIVE   | USER      |
|  0 | 4294566153 | ./#innodb_temp/temp_1.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566154 | ./#innodb_temp/temp_2.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566155 | ./#innodb_temp/temp_3.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566156 | ./#innodb_temp/temp_4.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566157 | ./#innodb_temp/temp_5.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566158 | ./#innodb_temp/temp_6.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566159 | ./#innodb_temp/temp_7.ibt  | 81920 | INACTIVE | NONE      |
|  0 | 4294566160 | ./#innodb_temp/temp_8.ibt  | 81920 | INACTIVE | NONE      |
```

```
+----+------------+---------------------------+-------+---------+-----------+
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.23 The INFORMATION_SCHEMA INNODB_TABLES Table

The `INNODB_TABLES` table provides metadata about `InnoDB` tables.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The `INNODB_TABLES` table has these columns:

- `TABLE_ID`

  An identifier for the `InnoDB` table. This value is unique across all databases in the instance.

- `NAME`

  The name of the table, preceded by the schema (database) name where appropriate (for example, `test/t1`). Names of databases and user tables are in the same case as they were originally defined, possibly influenced by the `lower_case_table_names` setting.

- `FLAG`

  A numeric value that represents bit-level information about table format and storage characteristics.

- `N_COLS`

  The number of columns in the table. The number reported includes three hidden columns that are created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`). The number reported also includes virtual generated columns, if present.

- `SPACE`

  An identifier for the tablespace where the table resides. 0 means the `InnoDB` system tablespace. Any other number represents either a file-per-table tablespace or a general tablespace. This identifier stays the same after a `TRUNCATE TABLE` statement. For file-per-table tablespaces, this identifier is unique for tables across all databases in the instance.

- `ROW_FORMAT`

  The table's row format (`Compact`, `Redundant`, `Dynamic`, or `Compressed`).

- `ZIP_PAGE_SIZE`

  The zip page size. Applies only to tables with a row format of `Compressed`.

- `SPACE_TYPE`

  The type of tablespace to which the table belongs. Possible values include `System` for the system tablespace, `General` for general tablespaces, and `Single` for file-per-table tablespaces. Tables assigned to the system tablespace using `CREATE TABLE` or `ALTER TABLE`

TABLESPACE=innodb_system have a SPACE_TYPE of General. For more information, see CREATE TABLESPACE.

- INSTANT_COLS

  The number of columns in the table prior to adding the first instant column using ALTER TABLE ... ADD COLUMN with ALGORITHM=INSTANT.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLES WHERE TABLE_ID = 214\G
*************************** 1. row ***************************
     TABLE_ID: 214
         NAME: test/t1
         FLAG: 129
       N_COLS: 4
        SPACE: 233
   ROW_FORMAT: Compact
ZIP_PAGE_SIZE: 0
   SPACE_TYPE: General
 INSTANT_COLS: 0
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

5.24 The INFORMATION_SCHEMA INNODB_TABLESPACES Table

The INNODB_TABLESPACES table provides metadata about InnoDB file-per-table, general, and undo tablespaces.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

> **Note**
>
> The INFORMATION_SCHEMA FILES table reports metadata for InnoDB tablespace types including file-per-table tablespaces, general tablespaces, the system tablespace, the global temporary tablespace, and undo tablespaces.

The INNODB_TABLESPACES table has these columns:

- SPACE

  The tablespace ID.

- NAME

  The schema (database) and table name.

- FLAG

  A numeric value that represents bit-level information about tablespace format and storage characteristics.

- ROW_FORMAT

The tablespace row format (`Compact or Redundant`, `Dynamic` or `Compressed`, or `Undo`). The data in this column is interpreted from the tablespace flag information that resides in the data file.

There is no way to determine from this flag information if the tablespace row format is `Redundant` or `Compact`, which is why one of the possible `ROW_FORMAT` values is `Compact or Redundant`.

- `PAGE_SIZE`

  The tablespace page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `ZIP_PAGE_SIZE`

  The tablespace zip page size. The data in this column is interpreted from the tablespace flags information that resides in the `.ibd` file.

- `SPACE_TYPE`

  The type of tablespace. Possible values include `General` for general tablespaces, `Single` for file-per-table tablespaces, `System` for the system tablespace, and `Undo` for undo tablespaces.

- `FS_BLOCK_SIZE`

  The file system block size, which is the unit size used for hole punching. This column pertains to the `InnoDB` transparent page compression feature.

- `FILE_SIZE`

  The apparent size of the file, which represents the maximum size of the file, uncompressed. This column pertains to the `InnoDB` transparent page compression feature.

- `ALLOCATED_SIZE`

  The actual size of the file, which is the amount of space allocated on disk. This column pertains to the `InnoDB` transparent page compression feature.

- `AUTOEXTEND_SIZE`

  The auto-extend size of the tablespace. This column was added in MySQL 8.0.23.

- `SERVER_VERSION`

  The MySQL version that created the tablespace, or the MySQL version into which the tablespace was imported, or the version of the last major MySQL version upgrade. The value is unchanged by a release series upgrade, such as an upgrade from MySQL 8.0.*x* to 8.0.*y*. The value can be considered a “creation” marker or “certified” marker for the tablespace.

- `SPACE_VERSION`

  The tablespace version, used to track changes to the tablespace format.

- `ENCRYPTION`

  Whether the tablespace is encrypted. This column was added in MySQL 8.0.13.

- `STATE`

  The tablespace state. This column was added in MySQL 8.0.14.

For file-per-table and general tablespaces, states include:

- `normal`: The tablespace is normal and active.
- `discarded`: The tablespace was discarded by an `ALTER TABLE ... DISCARD TABLESPACE` statement.
- `corrupted`: The tablespace is identified by `InnoDB` as corrupted.

For undo tablespaces, states include:

- `active`: Rollback segments in the undo tablespace can be allocated to new transactions.
- `inactive`: Rollback segments in the undo tablespace are no longer used by new transactions. The truncate process is in progress. The undo tablespace was either selected by the purge thread implicitly or was made inactive by an `ALTER UNDO TABLESPACE ... SET INACTIVE` statement.
- `empty`: The undo tablespace was truncated and is no longer active. It is ready to be dropped or made active again by an `ALTER UNDO TABLESPACE ... SET INACTIVE` statement.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES WHERE SPACE = 26\G
*************************** 1. row ***************************
         SPACE: 26
          NAME: test/t1
          FLAG: 0
    ROW_FORMAT: Compact or Redundant
     PAGE_SIZE: 16384
 ZIP_PAGE_SIZE: 0
    SPACE_TYPE: Single
 FS_BLOCK_SIZE: 4096
     FILE_SIZE: 98304
ALLOCATED_SIZE: 65536
AUTOEXTEND_SIZE: 0
SERVER_VERSION: 8.0.23
 SPACE_VERSION: 1
    ENCRYPTION: N
         STATE: normal
```

Notes

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.25 The INFORMATION_SCHEMA INNODB_TABLESPACES_BRIEF Table

The `INNODB_TABLESPACES_BRIEF` table provides space ID, name, path, flag, and space type metadata for file-per-table, general, undo, and system tablespaces.

`INNODB_TABLESPACES` provides the same metadata but loads more slowly because other metadata provided by the table, such as `FS_BLOCK_SIZE`, `FILE_SIZE`, and `ALLOCATED_SIZE`, must be loaded dynamically.

Space and path metadata is also provided by the `INNODB_DATAFILES` table.

The INNODB_TABLESPACES_BRIEF table has these columns:

- SPACE

  The tablespace ID.

- NAME

  The tablespace name. For file-per-table tablespaces, the name is in the form of *schema/table_name*.

- PATH

  The tablespace data file path. If a file-per-table tablespace is created in a location outside the MySQL data directory, the path value is a fully qualified directory path. Otherwise, the path is relative to the data directory.

- FLAG

  A numeric value that represents bit-level information about tablespace format and storage characteristics.

- SPACE_TYPE

  The type of tablespace. Possible values include General for InnoDB general tablespaces, Single for InnoDB file-per-table tablespaces, and System for the InnoDB system tablespace.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESPACES_BRIEF WHERE SPACE = 7;
+-------+---------+---------------+-------+------------+
| SPACE | NAME    | PATH          | FLAG  | SPACE_TYPE |
+-------+---------+---------------+-------+------------+
| 7     | test/t1 | ./test/t1.ibd | 16417 | Single     |
+-------+---------+---------------+-------+------------+
```

Notes

- You must have the PROCESS privilege to query this table.
- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

5.26 The INFORMATION_SCHEMA INNODB_TABLESTATS View

The INNODB_TABLESTATS table provides a view of low-level status information about InnoDB tables. This data is used by the MySQL optimizer to calculate which index to use when querying an InnoDB table. This information is derived from in-memory data structures rather than data stored on disk. There is no corresponding internal InnoDB system table.

InnoDB tables are represented in this view if they have been opened since the last server restart and have not aged out of the table cache. Tables for which persistent stats are available are always represented in this view.

Table statistics are updated only for DELETE or UPDATE operations that modify indexed columns. Statistics are not updated by operations that modify only nonindexed columns.

ANALYZE TABLE clears table statistics and sets the STATS_INITIALIZED column to Uninitialized. Statistics are collected again the next time the table is accessed.

For related usage information and examples, see InnoDB INFORMATION_SCHEMA Schema Object Tables.

The INNODB_TABLESTATS table has these columns:

- TABLE_ID

  An identifier representing the table for which statistics are available; the same value as INNODB_TABLES.TABLE_ID.

- NAME

  The name of the table; the same value as INNODB_TABLES.NAME.

- STATS_INITIALIZED

  The value is Initialized if the statistics are already collected, Uninitialized if not.

- NUM_ROWS

  The current estimated number of rows in the table. Updated after each DML operation. The value could be imprecise if uncommitted transactions are inserting into or deleting from the table.

- CLUST_INDEX_SIZE

  The number of pages on disk that store the clustered index, which holds the InnoDB table data in primary key order. This value might be null if no statistics are collected yet for the table.

- OTHER_INDEX_SIZE

  The number of pages on disk that store all secondary indexes for the table. This value might be null if no statistics are collected yet for the table.

- MODIFIED_COUNTER

  The number of rows modified by DML operations, such as INSERT, UPDATE, DELETE, and also cascade operations from foreign keys. This column is reset each time table statistics are recalculated

- AUTOINC

  The next number to be issued for any auto-increment-based operation. The rate at which the AUTOINC value changes depends on how many times auto-increment numbers have been requested and how many numbers are granted per request.

- REF_COUNT

  When this counter reaches zero, the table metadata can be evicted from the table cache.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TABLESTATS where TABLE_ID = 71\G
*************************** 1. row ***************************
        TABLE_ID: 71
            NAME: test/t1
STATS_INITIALIZED: Initialized
        NUM_ROWS: 1
 CLUST_INDEX_SIZE: 1
 OTHER_INDEX_SIZE: 0
 MODIFIED_COUNTER: 1
          AUTOINC: 0
```

```
        REF_COUNT: 1
```

Notes

- This table is useful primarily for expert-level performance monitoring, or when developing performance-related extensions for MySQL.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.27 The INFORMATION_SCHEMA INNODB_TEMP_TABLE_INFO Table

The `INNODB_TEMP_TABLE_INFO` table provides information about user-created `InnoDB` temporary tables that are active in an `InnoDB` instance. It does not provide information about internal `InnoDB` temporary tables used by the optimizer. The `INNODB_TEMP_TABLE_INFO` table is created when first queried, exists only in memory, and is not persisted to disk.

For usage information and examples, see InnoDB INFORMATION_SCHEMA Temporary Table Info Table.

The `INNODB_TEMP_TABLE_INFO` table has these columns:

- `TABLE_ID`

  The table ID of the temporary table.

- `NAME`

  The name of the temporary table.

- `N_COLS`

  The number of columns in the temporary table. The number includes three hidden columns created by `InnoDB` (`DB_ROW_ID`, `DB_TRX_ID`, and `DB_ROLL_PTR`).

- `SPACE`

  The ID of the temporary tablespace where the temporary table resides.

Example

```
mysql> CREATE TEMPORARY TABLE t1 (c1 INT PRIMARY KEY) ENGINE=INNODB;
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TEMP_TABLE_INFO\G
*************************** 1. row ***************************
TABLE_ID: 97
    NAME: #sql8c88_43_0
  N_COLS: 4
   SPACE: 76
```

Notes

- This table is useful primarily for expert-level monitoring.
- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

5.28 The INFORMATION_SCHEMA INNODB_TRX Table

The `INNODB_TRX` table provides information about every transaction currently executing inside `InnoDB`, including whether the transaction is waiting for a lock, when the transaction started, and the SQL statement the transaction is executing, if any.

For usage information, see Using InnoDB Transaction and Locking Information.

The `INNODB_TRX` table has these columns:

- `TRX_ID`

  A unique transaction ID number, internal to `InnoDB`. These IDs are not created for transactions that are read only and nonlocking. For details, see Optimizing InnoDB Read-Only Transactions.

- `TRX_WEIGHT`

  The weight of a transaction, reflecting (but not necessarily the exact count of) the number of rows altered and the number of rows locked by the transaction. To resolve a deadlock, `InnoDB` selects the transaction with the smallest weight as the “victim” to roll back. Transactions that have changed nontransactional tables are considered heavier than others, regardless of the number of altered and locked rows.

- `TRX_STATE`

  The transaction execution state. Permitted values are `RUNNING`, `LOCK WAIT`, `ROLLING BACK`, and `COMMITTING`.

- `TRX_STARTED`

  The transaction start time.

- `TRX_REQUESTED_LOCK_ID`

  The ID of the lock the transaction is currently waiting for, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`. To obtain details about the lock, join this column with the `ENGINE_LOCK_ID` column of the Performance Schema `data_locks` table.

- `TRX_WAIT_STARTED`

  The time when the transaction started waiting on the lock, if `TRX_STATE` is `LOCK WAIT`; otherwise `NULL`.

- `TRX_MYSQL_THREAD_ID`

  The MySQL thread ID. To obtain details about the thread, join this column with the `ID` column of the `INFORMATION_SCHEMA PROCESSLIST` table, but see Persistence and Consistency of InnoDB Transaction and Locking Information.

- `TRX_QUERY`

  The SQL statement that is being executed by the transaction.

- `TRX_OPERATION_STATE`

  The transaction's current operation, if any; otherwise `NULL`.

- `TRX_TABLES_IN_USE`

The number of InnoDB tables used while processing the current SQL statement of this transaction.

- TRX_TABLES_LOCKED

  The number of InnoDB tables that the current SQL statement has row locks on. (Because these are row locks, not table locks, the tables can usually still be read from and written to by multiple transactions, despite some rows being locked.)

- TRX_LOCK_STRUCTS

  The number of locks reserved by the transaction.

- TRX_LOCK_MEMORY_BYTES

  The total size taken up by the lock structures of this transaction in memory.

- TRX_ROWS_LOCKED

  The approximate number or rows locked by this transaction. The value might include delete-marked rows that are physically present but not visible to the transaction.

- TRX_ROWS_MODIFIED

  The number of modified and inserted rows in this transaction.

- TRX_CONCURRENCY_TICKETS

  A value indicating how much work the current transaction can do before being swapped out, as specified by the innodb_concurrency_tickets system variable.

- TRX_ISOLATION_LEVEL

  The isolation level of the current transaction.

- TRX_UNIQUE_CHECKS

  Whether unique checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- TRX_FOREIGN_KEY_CHECKS

  Whether foreign key checks are turned on or off for the current transaction. For example, they might be turned off during a bulk data load.

- TRX_LAST_FOREIGN_KEY_ERROR

  The detailed error message for the last foreign key error, if any; otherwise NULL.

- TRX_ADAPTIVE_HASH_LATCHED

  Whether the adaptive hash index is locked by the current transaction. When the adaptive hash index search system is partitioned, a single transaction does not lock the entire adaptive hash index. Adaptive hash index partitioning is controlled by innodb_adaptive_hash_index_parts, which is set to 8 by default.

- TRX_ADAPTIVE_HASH_TIMEOUT

  Whether to relinquish the search latch immediately for the adaptive hash index, or reserve it across calls from MySQL. When there is no adaptive hash index contention, this value remains zero and statements

reserve the latch until they finish. During times of contention, it counts down to zero, and statements release the latch immediately after each row lookup. When the adaptive hash index search system is partitioned (controlled by `innodb_adaptive_hash_index_parts`), the value remains 0.

- `TRX_IS_READ_ONLY`

  A value of 1 indicates the transaction is read only.

- `TRX_AUTOCOMMIT_NON_LOCKING`

  A value of 1 indicates the transaction is a `SELECT` statement that does not use the `FOR UPDATE` or `LOCK IN SHARED MODE` clauses, and is executing with `autocommit` enabled so that the transaction contains only this one statement. When this column and `TRX_IS_READ_ONLY` are both 1, `InnoDB` optimizes the transaction to reduce the overhead associated with transactions that change table data.

- `TRX_SCHEDULE_WEIGHT`

  The transaction schedule weight assigned by the Contention-Aware Transaction Scheduling (CATS) algorithm to transactions waiting for a lock. The value is relative to the values of other transactions. A higher value has a greater weight. A value is computed only for transactions in a `LOCK WAIT` state, as reported by the `TRX_STATE` column. A NULL value is reported for transactions that are not waiting for a lock. The `TRX_SCHEDULE_WEIGHT` value is different from the `TRX_WEIGHT` value, which is computed by a different algorithm for a different purpose.

Example

```
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_TRX\G
*************************** 1. row ***************************
                    trx_id: 1510
                 trx_state: RUNNING
               trx_started: 2014-11-19 13:24:40
     trx_requested_lock_id: NULL
          trx_wait_started: NULL
                trx_weight: 586739
       trx_mysql_thread_id: 2
                 trx_query: DELETE FROM employees.salaries WHERE salary > 65000
       trx_operation_state: updating or deleting
         trx_tables_in_use: 1
         trx_tables_locked: 1
          trx_lock_structs: 3003
     trx_lock_memory_bytes: 450768
           trx_rows_locked: 1407513
         trx_rows_modified: 583736
   trx_concurrency_tickets: 0
       trx_isolation_level: REPEATABLE READ
         trx_unique_checks: 1
    trx_foreign_key_checks: 1
trx_last_foreign_key_error: NULL
 trx_adaptive_hash_latched: 0
 trx_adaptive_hash_timeout: 10000
          trx_is_read_only: 0
trx_autocommit_non_locking: 0
       trx_schedule_weight: NULL
```

Notes

- Use this table to help diagnose performance problems that occur during times of heavy concurrent load. Its contents are updated as described in Persistence and Consistency of InnoDB Transaction and Locking Information.

- You must have the `PROCESS` privilege to query this table.

- Use the INFORMATION_SCHEMA COLUMNS table or the SHOW COLUMNS statement to view additional information about the columns of this table, including data types and default values.

5.29 The INFORMATION_SCHEMA INNODB_VIRTUAL Table

The INNODB_VIRTUAL table provides metadata about InnoDB virtual generated columns and columns upon which virtual generated columns are based.

A row appears in the INNODB_VIRTUAL table for each column upon which a virtual generated column is based.

The INNODB_VIRTUAL table has these columns:

- TABLE_ID

  An identifier representing the table associated with the virtual column; the same value as INNODB_TABLES.TABLE_ID.

- POS

  The position value of the virtual generated column. The value is large because it encodes the column sequence number and ordinal position. The formula used to calculate the value uses a bitwise operation:

```
((nth virtual generated column for the InnoDB instance + 1) << 16)
+ the ordinal position of the virtual generated column
```

  For example, if the first virtual generated column in the InnoDB instance is the third column of the table, the formula is (0 + 1) << 16) + 2. The first virtual generated column in the InnoDB instance is always number 0. As the third column in the table, the ordinal position of the virtual generated column is 2. Ordinal positions are counted from 0.

- BASE_POS

  The ordinal position of the columns upon which a virtual generated column is based.

Example

```
mysql> CREATE TABLE `t1` (
         `a` int(11) DEFAULT NULL,
         `b` int(11) DEFAULT NULL,
         `c` int(11) GENERATED ALWAYS AS (a+b) VIRTUAL,
         `h` varchar(10) DEFAULT NULL
       ) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
mysql> SELECT * FROM INFORMATION_SCHEMA.INNODB_VIRTUAL
       WHERE TABLE_ID IN
         (SELECT TABLE_ID FROM INFORMATION_SCHEMA.INNODB_TABLES
          WHERE NAME LIKE "test/t1");
+----------+-------+----------+
| TABLE_ID | POS   | BASE_POS |
+----------+-------+----------+
|       98 | 65538 |        0 |
|       98 | 65538 |        1 |
+----------+-------+----------+
```

Notes

- If a constant value is assigned to a virtual generated column, as in the following table, an entry for the column does not appear in the INNODB_VIRTUAL table. For an entry to appear, a virtual generated column must have a base column.

```
CREATE TABLE `t1` (
  `a` int(11) DEFAULT NULL,
  `b` int(11) DEFAULT NULL,
  `c` int(11) GENERATED ALWAYS AS (5) VIRTUAL
) ENGINE=InnoDB DEFAULT CHARSET=utf8mb4;
```

However, metadata for such a column does appear in the `INNODB_COLUMNS` table.

- You must have the `PROCESS` privilege to query this table.
- Use the `INFORMATION_SCHEMA COLUMNS` table or the `SHOW COLUMNS` statement to view additional information about the columns of this table, including data types and default values.

Chapter 6 INFORMATION_SCHEMA Thread Pool Tables

Table of Contents

> **Note**
>
> As of MySQL 8.0.14, the `INFORMATION_SCHEMA` thread pool tables are also available as Performance Schema tables. (See Performance Schema Thread Pool Tables.) The `INFORMATION_SCHEMA` tables are deprecated; expect them be removed in a future version of MySQL. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:
>
> ```
> SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
> ```
>
> The application should use this query instead:
>
> ```
> SELECT * FROM performance_schema.tp_thread_state;
> ```

The following sections describe the `INFORMATION_SCHEMA` tables associated with the thread pool plugin (see MySQL Enterprise Thread Pool). They provide information about thread pool operation:

- `TP_THREAD_GROUP_STATE`: Information about thread pool thread group states
- `TP_THREAD_GROUP_STATS`: Thread group statistics
- `TP_THREAD_STATE`: Information about thread pool thread states

Rows in these tables represent snapshots in time. In the case of `TP_THREAD_STATE`, all rows for a thread group comprise a snapshot in time. Thus, the MySQL server holds the mutex of the thread group while producing the snapshot. But it does not hold mutexes on all thread groups at the same time, to prevent a statement against `TP_THREAD_STATE` from blocking the entire MySQL server.

The `INFORMATION_SCHEMA` thread pool tables are implemented by individual plugins and the decision whether to load one can be made independently of the others (see Thread Pool Installation). However, the content of all the tables depends on the thread pool plugin being enabled. If a table plugin is enabled but the thread pool plugin is not, the table becomes visible and can be accessed but is empty.

6.1 INFORMATION_SCHEMA Thread Pool Table Reference

The following table summarizes `INFORMATION_SCHEMA` thread pool tables. For greater detail, see the individual table descriptions.

Table 6.1 INFORMATION_SCHEMA Thread Pool Tables

Table Name	Description
`TP_THREAD_GROUP_STATE`	Thread pool thread group states
`TP_THREAD_GROUP_STATS`	Thread pool thread group statistics
`TP_THREAD_STATE`	Thread pool thread information

6.2 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATE Table

> **Note**
>
> As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Performance Schema Thread Pool Tables.) The `INFORMATION_SCHEMA` tables are deprecated; expect them to be removed in a future version of MySQL. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:
>
> ```
> SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATE;
> ```
>
> The application should use this query instead:
>
> ```
> SELECT * FROM performance_schema.tp_thread_group_state;
> ```

The `TP_THREAD_GROUP_STATE` table has one row per thread group in the thread pool. Each row provides information about the current state of a group.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_GROUP_STATE` table, see The tp_thread_group_state Table. The Performance Schema `tp_thread_group_state` table has equivalent columns.

6.3 The INFORMATION_SCHEMA TP_THREAD_GROUP_STATS Table

> **Note**
>
> As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Performance Schema Thread Pool Tables.) The `INFORMATION_SCHEMA` tables are deprecated; expect them to be removed in a future version of MySQL. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:
>
> ```
> SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_GROUP_STATS;
> ```
>
> The application should use this query instead:
>
> ```
> SELECT * FROM performance_schema.tp_thread_group_stats;
> ```

The `TP_THREAD_GROUP_STATS` table reports statistics per thread group. There is one row per group.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_GROUP_STATS` table, see The tp_thread_group_stats Table. The Performance Schema `tp_thread_group_stats` table has equivalent columns.

6.4 The INFORMATION_SCHEMA TP_THREAD_STATE Table

> **Note**
>
> As of MySQL 8.0.14, the thread pool `INFORMATION_SCHEMA` tables are also available as Performance Schema tables. (See Performance Schema Thread Pool Tables.) The `INFORMATION_SCHEMA` tables are deprecated; expect them to be

> removed in a future version of MySQL. Applications should transition away from the old tables to the new tables. For example, if an application uses this query:
>
> ```
> SELECT * FROM INFORMATION_SCHEMA.TP_THREAD_STATE;
> ```
>
> The application should use this query instead:
>
> ```
> SELECT * FROM performance_schema.tp_thread_state;
> ```

The `TP_THREAD_STATE` table has one row per thread created by the thread pool to handle connections.

For descriptions of the columns in the `INFORMATION_SCHEMA TP_THREAD_STATE` table, see The tp_thread_state Table. The Performance Schema `tp_thread_state` table has equivalent columns.

Chapter 7 INFORMATION_SCHEMA Connection-Control Tables

Table of Contents

The following sections describe the `INFORMATION_SCHEMA` tables associated with the `CONNECTION_CONTROL` plugin.

7.1 INFORMATION_SCHEMA Connection-Control Table Reference

The following table summarizes `INFORMATION_SCHEMA` connection-control tables. For greater detail, see the individual table descriptions.

Table 7.1 INFORMATION_SCHEMA Connection-Control Tables

Table Name	Description
`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS`	Current number of consecutive failed connection attempts per account

7.2 The INFORMATION_SCHEMA CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS Table

This table provides information about the current number of consecutive failed connection attempts per account (user/host combination).

`CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` has these columns:

- `USERHOST`

  The user/host combination indicating an account that has failed connection attempts, in *`'user_name'@'host_name'`* format.

- `FAILED_ATTEMPTS`

  The current number of consecutive failed connection attempts for the `USERHOST` value. This counts all failed attempts, regardless of whether they were delayed. The number of attempts for which the server added a delay to its response is the difference between the `FAILED_ATTEMPTS` value and the `connection_control_failed_connections_threshold` system variable value.

Notes

- The `CONNECTION_CONTROL_FAILED_LOGIN_ATTEMPTS` plugin must be activated for this table to be available, and the `CONNECTION_CONTROL` plugin must be activated or the table contents are always empty. See The Connection-Control Plugins.
- The table contains rows only for accounts that have had one or more consecutive failed connection attempts without a subsequent successful attempt. When an account connects successfully, its failed-connection count is reset to zero and the server removes any row corresponding to the account.

- Assigning a value to the `connection_control_failed_connections_threshold` system variable at runtime resets all accumulated failed-connection counters to zero, which causes the table to become empty.

Chapter 8 INFORMATION_SCHEMA MySQL Enterprise Firewall Tables

Table of Contents

The following sections describe the `INFORMATION_SCHEMA` tables associated with MySQL Enterprise Firewall (see MySQL Enterprise Firewall). They provide views into the firewall in-memory data cache. These tables are available only if the appropriate firewall plugins are enabled.

8.1 INFORMATION_SCHEMA Firewall Table Reference

The following table summarizes `INFORMATION_SCHEMA` firewall tables. For greater detail, see the individual table descriptions.

Table 8.1 INFORMATION_SCHEMA Firewall Tables

Table Name	Description	Deprecated
`MYSQL_FIREWALL_USERS`	Firewall in-memory data for account profiles	8.0.26
`MYSQL_FIREWALL_WHITELIST`	Firewall in-memory data for account profile allowlists	8.0.26

8.2 The INFORMATION_SCHEMA MYSQL_FIREWALL_USERS Table

The `MYSQL_FIREWALL_USERS` table provides a view into the in-memory data cache for MySQL Enterprise Firewall. It lists names and operational modes of registered firewall account profiles. It is used in conjunction with the `mysql.firewall_users` system table that provides persistent storage of firewall data; see MySQL Enterprise Firewall Tables.

The `MYSQL_FIREWALL_USERS` table has these columns:

- `USERHOST`

  The account profile name. Each account name has the format *`user_name@host_name`*.

- `MODE`

  The current operational mode for the profile. Permitted mode values are `OFF`, `DETECTING`, `PROTECTING`, `RECORDING`, and `RESET`. For details about their meanings, see Firewall Concepts.

As of MySQL 8.0.26, this table is deprecated and subject to removal in a future MySQL version. See Migrating Account Profiles to Group Profiles.

8.3 The INFORMATION_SCHEMA MYSQL_FIREWALL_WHITELIST Table

The `MYSQL_FIREWALL_WHITELIST` table provides a view into the in-memory data cache for MySQL Enterprise Firewall. It lists allowlist rules of registered firewall account profiles. It is used in conjunction with the `mysql.firewall_whitelist` system table that provides persistent storage of firewall data; see MySQL Enterprise Firewall Tables.

The `MYSQL_FIREWALL_WHITELIST` table has these columns:

- `USERHOST`

  The account profile name. Each account name has the format *`user_name@host_name`*.

- `RULE`

  A normalized statement indicating an acceptable statement pattern for the profile. A profile allowlist is the union of its rules.

As of MySQL 8.0.26, this table is deprecated and subject to removal in a future MySQL version. See Migrating Account Profiles to Group Profiles.

Chapter 9 Extensions to SHOW Statements

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:

```
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:

```
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:

```
mysql> SHOW CHARACTER SET;
+----------+-----------------------------+---------------------+--------+
| Charset  | Description                 | Default collation   | Maxlen |
+----------+-----------------------------+---------------------+--------+
| big5     | Big5 Traditional Chinese    | big5_chinese_ci     |      2 |
| dec8     | DEC West European           | dec8_swedish_ci     |      1 |
| cp850    | DOS West European           | cp850_general_ci    |      1 |
| hp8      | HP West European            | hp8_english_ci      |      1 |
| koi8r    | KOI8-R Relcom Russian       | koi8r_general_ci    |      1 |
| latin1   | cp1252 West European        | latin1_swedish_ci   |      1 |
| latin2   | ISO 8859-2 Central European | latin2_general_ci   |      1 |
...
```

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%';
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

This statement displays the multibyte character sets:

```
mysql> SHOW CHARACTER SET WHERE Maxlen > 1;
+---------+---------------------------+---------------------+--------+
| Charset | Description               | Default collation   | Maxlen |
+---------+---------------------------+---------------------+--------+
| big5    | Big5 Traditional Chinese  | big5_chinese_ci     |      2 |
| ujis    | EUC-JP Japanese           | ujis_japanese_ci    |      3 |
| sjis    | Shift-JIS Japanese        | sjis_japanese_ci    |      2 |
| euckr   | EUC-KR Korean             | euckr_korean_ci     |      2 |
| gb2312  | GB2312 Simplified Chinese | gb2312_chinese_ci   |      2 |
| gbk     | GBK Simplified Chinese    | gbk_chinese_ci      |      2 |
| utf8    | UTF-8 Unicode             | utf8_general_ci     |      3 |
| ucs2    | UCS-2 Unicode             | ucs2_general_ci     |      2 |
| cp932   | SJIS for Windows Japanese | cp932_japanese_ci   |      2 |
| eucjpms | UJIS for Windows Japanese | eucjpms_japanese_ci |      3 |
+---------+---------------------------+---------------------+--------+
```

Chapter 10 MySQL 8.0 FAQ: INFORMATION_SCHEMA

Questions

Questions and Answers

10.1: Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?

See Chapter 1, *INFORMATION_SCHEMA Tables*

10.2: Is there a discussion forum for `INFORMATION_SCHEMA`?

See https://forums.mysql.com/list.php?101.

10.3: 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`.

10.4: 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.

10.5: 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*.