MySQL Information Schema

Abstract

This is the MySQL Information Schema extract from the MySQL 5.1 Reference Manual.

For legal information, see the Legal Notices.

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

For additional documentation on MySQL products, including translations of the documentation into other languages, and downloadable versions in variety of formats, including HTML and PDF formats, see the MySQL Documentation Library.

Document generated on: 2016-08-16 (revision: 48562)

Table of Contents

Preface and Legal Notices

This is the MySQL Information Schema extract from the MySQL 5.1 Reference Manual.

Legal Notices

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*.

Usage Notes for the INFORMATION_SCHEMA Database

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.

Example

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 Collation and 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 27, *Extensions to SHOW Statements*.

Privileges

Each MySQL user has the right to access these tables, 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`. These restrictions do not apply for `InnoDB` tables; you can see them with only 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 INFORMATION_SCHEMA Optimization.

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 column is the `ENGINE` column in the `INFORMATION_SCHEMA.TABLES` table.

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.

For information about `INFORMATION_SCHEMA` tables specific to the `NDB` storage engine (MySQL Cluster), see INFORMATION_SCHEMA Tables for MySQL Cluster.

For answers to questions that are often asked concerning the `INFORMATION_SCHEMA` database, see Chapter 28, *MySQL 5.1 FAQ: INFORMATION_SCHEMA*.

Chapter 2 The INFORMATION_SCHEMA SCHEMATA Table

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

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CATALOG_NAME		NULL
SCHEMA_NAME	Database	
DEFAULT_CHARACTER_SET_NAME		
DEFAULT_COLLATION_NAME		
SQL_PATH		NULL

The following statements are equivalent:

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

Chapter 3 The INFORMATION_SCHEMA TABLES Table

The TABLES table provides information about tables in databases.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		NULL
TABLE_SCHEMA	Table_...	
TABLE_NAME	Table_...	
TABLE_TYPE		
ENGINE	Engine	MySQL extension
VERSION	Version	The version number of the table's .frm file, MySQL extension
ROW_FORMAT	Row_format	MySQL extension
TABLE_ROWS	Rows	MySQL extension
AVG_ROW_LENGTH	Avg_row_length	MySQL extension
DATA_LENGTH	Data_length	MySQL extension
MAX_DATA_LENGTH	Max_data_length	MySQL extension
INDEX_LENGTH	Index_length	MySQL extension
DATA_FREE	Data_free	MySQL extension
AUTO_INCREMENT	Auto_increment	MySQL extension
CREATE_TIME	Create_time	MySQL extension
UPDATE_TIME	Update_time	MySQL extension
CHECK_TIME	Check_time	MySQL extension
TABLE_COLLATION	Collation	MySQL extension
CHECKSUM	Checksum	MySQL extension
CREATE_OPTIONS	Create_options	MySQL extension
TABLE_COMMENT	Comment	MySQL extension
INDEX_COMMENT	Index_comment	MySQL extension

Notes:

- Refer to SHOW TABLE STATUS for field descriptions.
- TABLE_SCHEMA and TABLE_NAME are a single field in a SHOW display, for example Table_in_db1.
- TABLE_TYPE should be BASE TABLE or VIEW. The TABLES table does not list TEMPORARY tables.
- For partitioned tables, beginning with MySQL 5.1.9, the ENGINE column shows the name of the storage engine used by all partitions. (Previously, this column showed PARTITION for such tables.)
- The TABLE_ROWS column is NULL if the table is in the INFORMATION_SCHEMA database.

  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.)

- Prior to MySQL 5.1.12, MySQL Cluster allocated storage for variable-width columns in 10-page extents of 32 kilobytes each; thus, the `DATA_LENGTH` for such columns was reported in increments of 320 KB. Beginning with MySQL 5.1.12, the `DATA_LENGTH` column reflects the true amount of storage for variable-width columns of `NDB` tables. (Bug #18413)

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

- Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, for MySQL 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.)

- Beginning with MySQL 5.1.28, the `DATA_FREE` column shows the free space in bytes for `InnoDB` tables.

  Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, `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 `InnoDB` tables, the `CREATE_TIME`, `UPDATE_TIME`, and `CHECK_TIME` columns are always `NULL`.

- We have nothing for the table's default character set. `TABLE_COLLATION` is close, because collation names begin with a character set name.

- Beginning with MySQL 5.1.9, the `CREATE_OPTIONS` column shows `partitioned` if the table is partitioned.

The following statements are equivalent:

```
SELECT table_name FROM INFORMATION_SCHEMA.TABLES
  WHERE table_schema = 'db_name'
  [AND table_name LIKE 'wild']
SHOW TABLES
  FROM db_name
  [LIKE 'wild']
```

Chapter 4 The INFORMATION_SCHEMA COLUMNS Table

The COLUMNS table provides information about columns in tables.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME	Field	
ORDINAL_POSITION		see notes
COLUMN_DEFAULT	Default	
IS_NULLABLE	Null	
DATA_TYPE	Type	
CHARACTER_MAXIMUM_LENGTH	Type	
CHARACTER_OCTET_LENGTH		
NUMERIC_PRECISION	Type	
NUMERIC_SCALE	Type	
CHARACTER_SET_NAME		
COLLATION_NAME	Collation	
COLUMN_TYPE	Type	MySQL extension
COLUMN_KEY	Key	MySQL extension
EXTRA	Extra	MySQL extension
PRIVILEGES	Privileges	MySQL extension
COLUMN_COMMENT	Comment	MySQL extension

Notes:

- In SHOW, the Type display includes values from several different COLUMNS columns.
- ORDINAL_POSITION is necessary because you might want to say ORDER BY ORDINAL_POSITION. Unlike SHOW, SELECT does not have automatic ordering.
- CHARACTER_OCTET_LENGTH should be the same as CHARACTER_MAXIMUM_LENGTH, except for multibyte character sets.
- CHARACTER_SET_NAME can be derived from Collation. For example, if you say SHOW FULL COLUMNS FROM t, and you see in the Collation column a value of latin1_swedish_ci, the character set is what is before the first underscore: latin1.

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']
```

Chapter 5 The INFORMATION_SCHEMA STATISTICS Table

The STATISTICS table provides information about table indexes.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		NULL
TABLE_SCHEMA		= Database
TABLE_NAME	Table	
NON_UNIQUE	Non_unique	
INDEX_SCHEMA		= Database
INDEX_NAME	Key_name	
SEQ_IN_INDEX	Seq_in_index	
COLUMN_NAME	Column_name	
COLLATION	Collation	
CARDINALITY	Cardinality	
SUB_PART	Sub_part	MySQL extension
PACKED	Packed	MySQL extension
NULLABLE	Null	MySQL extension
INDEX_TYPE	Index_type	MySQL extension
COMMENT	Comment	MySQL extension

Notes:

- There is no standard table for indexes. The preceding list is similar to what SQL Server 2000 returns for sp_statistics, except that we replaced the name QUALIFIER with CATALOG and we replaced the name OWNER with SCHEMA.

  Clearly, the preceding table and the output from SHOW INDEX are derived from the same parent. So the correlation is already close.

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
```

Chapter 6 The INFORMATION_SCHEMA USER_PRIVILEGES Table

The USER_PRIVILEGES table provides information about global privileges. This information comes from the mysql.user grant table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
GRANTEE		'*user_name*'@'*host_name*' value, MySQL extension
TABLE_CATALOG		NULL, MySQL extension
PRIVILEGE_TYPE		MySQL extension
IS_GRANTABLE		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the mysql.user table.

Chapter 7 The INFORMATION_SCHEMA SCHEMA_PRIVILEGES Table

The `SCHEMA_PRIVILEGES` table provides information about schema (database) privileges. This information comes from the `mysql.db` grant table.

`INFORMATION_SCHEMA` **Name**	**`SHOW`** **Name**	**Remarks**
`GRANTEE`		`'`*`user_name`*`'@'`*`host_name`*`'` value, MySQL extension
`TABLE_CATALOG`		`NULL`, MySQL extension
`TABLE_SCHEMA`		MySQL extension
`PRIVILEGE_TYPE`		MySQL extension
`IS_GRANTABLE`		MySQL extension

Notes:

- This is a nonstandard table. It takes its values from the `mysql.db` table.

Chapter 8 The INFORMATION_SCHEMA TABLE_PRIVILEGES Table

The TABLE_PRIVILEGES table provides information about table privileges. This information comes from the mysql.tables_priv grant table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
GRANTEE		'*user_name*'@'*host_name*' value
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES, ALTER, INDEX, DROP, CREATE VIEW.

The following statements are *not* equivalent:

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

Chapter 9 The INFORMATION_SCHEMA COLUMN_PRIVILEGES Table

The COLUMN_PRIVILEGES table provides information about column privileges. This information comes from the mysql.columns_priv grant table.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
GRANTEE		*'user_name'@'host_name'* value
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
PRIVILEGE_TYPE		
IS_GRANTABLE		

Notes:

- In the output from SHOW FULL COLUMNS, the privileges are all in one field and in lowercase, for example, select,insert,update,references. In COLUMN_PRIVILEGES, there is one privilege per row, in uppercase.
- PRIVILEGE_TYPE can contain one (and only one) of these values: SELECT, INSERT, UPDATE, REFERENCES.
- If the user has GRANT OPTION privilege, IS_GRANTABLE should be YES. Otherwise, IS_GRANTABLE should be NO. The output does not list GRANT OPTION as a separate privilege.

The following statements are *not* equivalent:

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

Chapter 10 The INFORMATION_SCHEMA CHARACTER_SETS Table

The CHARACTER_SETS table provides information about available character sets.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CHARACTER_SET_NAME	Charset	
DEFAULT_COLLATE_NAME	Default collation	
DESCRIPTION	Description	MySQL extension
MAXLEN	Maxlen	MySQL extension

The following statements are equivalent:

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

Chapter 11 The INFORMATION_SCHEMA COLLATIONS Table

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

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
COLLATION_NAME	Collation	
CHARACTER_SET_NAME	Charset	MySQL extension
ID	Id	MySQL extension
IS_DEFAULT	Default	MySQL extension
IS_COMPILED	Compiled	MySQL extension
SORTLEN	Sortlen	MySQL extension

- COLLATION_NAME is the collation name.
- CHARACTER_SET_NAME is the name of the character set with which the collation is associated.
- ID is the collation ID.
- IS_DEFAULT indicates whether the collation is the default for its character set.
- IS_COMPILED indicates whether the character set is compiled into the server.
- SORTLEN is related to the amount of memory required to sort strings expressed in the character set.

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

```
SELECT COLLATION_NAME FROM INFORMATION_SCHEMA.COLLATIONS
  [WHERE COLLATION_NAME LIKE 'wild']
SHOW COLLATION
  [LIKE 'wild']
```

Chapter 12 The INFORMATION_SCHEMA COLLATION_CHARACTER_SET_APPLICABILITY Table

The `COLLATION_CHARACTER_SET_APPLICABILITY` table indicates what character set is applicable for what collation. The columns are equivalent to the first two display fields that we get from `SHOW COLLATION`.

`INFORMATION_SCHEMA` **Name**	**`SHOW`** **Name**	**Remarks**
`COLLATION_NAME`	`Collation`	
`CHARACTER_SET_NAME`	`Charset`	

Chapter 13 The INFORMATION_SCHEMA TABLE_CONSTRAINTS Table

The TABLE_CONSTRAINTS table describes which tables have constraints.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_SCHEMA		
TABLE_NAME		
CONSTRAINT_TYPE		

Notes:

- The CONSTRAINT_TYPE value can be UNIQUE, PRIMARY KEY, or FOREIGN KEY.
- The UNIQUE and PRIMARY KEY information is about the same as what you get from the Key_name field in the output from SHOW INDEX when the Non_unique field is 0.
- The CONSTRAINT_TYPE column can contain one of these values: UNIQUE, PRIMARY KEY, FOREIGN KEY, CHECK. This is a CHAR (not ENUM) column. The CHECK value is not available until we support CHECK.

Chapter 14 The INFORMATION_SCHEMA KEY_COLUMN_USAGE Table

The KEY_COLUMN_USAGE table describes which key columns have constraints.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
TABLE_CATALOG		
TABLE_SCHEMA		
TABLE_NAME		
COLUMN_NAME		
ORDINAL_POSITION		
POSITION_IN_UNIQUE_CONSTRAINT		
REFERENCED_TABLE_SCHEMA		
REFERENCED_TABLE_NAME		
REFERENCED_COLUMN_NAME		

Notes:

- If the constraint is a foreign key, then this is the column of the foreign key, not the column that the foreign key references.
- The value of ORDINAL_POSITION is the column's position within the constraint, not the column's position within the table. Column positions are numbered beginning with 1.
- The value of POSITION_IN_UNIQUE_CONSTRAINT is NULL for unique and primary-key constraints. For foreign-key constraints, it is the ordinal position in key of the table that is being referenced.

  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.

- One row with `CONSTRAINT_NAME` = `'CO'`, `TABLE_NAME` = `'t3'`, `COLUMN_NAME` = `'s2'`, `ORDINAL_POSITION` = `1`, `POSITION_IN_UNIQUE_CONSTRAINT` = `1`.

Chapter 15 The INFORMATION_SCHEMA ROUTINES Table

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

The column named “mysql.proc name” indicates the mysql.proc table column that corresponds to the INFORMATION_SCHEMA.ROUTINES table column, if any.

INFORMATION_SCHEMA Name	**mysql.proc Name**	**Remarks**
SPECIFIC_NAME	specific_name	
ROUTINE_CATALOG		NULL
ROUTINE_SCHEMA	db	
ROUTINE_NAME	name	
ROUTINE_TYPE	type	{PROCEDURE|FUNCTION}
DTD_IDENTIFIER		data type descriptor
ROUTINE_BODY		SQL
ROUTINE_DEFINITION	body or body_utf8	see Notes
EXTERNAL_NAME		NULL
EXTERNAL_LANGUAGE	language	NULL
PARAMETER_STYLE		SQL
IS_DETERMINISTIC	is_deterministic	
SQL_DATA_ACCESS	sql_data_access	
SQL_PATH		NULL
SECURITY_TYPE	security_type	
CREATED	created	
LAST_ALTERED	modified	
SQL_MODE	sql_mode	MySQL extension
ROUTINE_COMMENT	comment	MySQL extension
DEFINER	definer	MySQL extension
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension
DATABASE_COLLATION		MySQL extension

Notes:

- MySQL calculates EXTERNAL_LANGUAGE thus:
  - If mysql.proc.language='SQL', EXTERNAL_LANGUAGE is NULL
  - Otherwise, EXTERNAL_LANGUAGE is what is in mysql.proc.language. However, we do not have external languages yet, so it is always NULL.
- ROUTINE_DEFINITION is what is in mysql.proc.body_utf8 as of MySQL 5.1.21, mysql.proc.body before 5.1.21.
- 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 column holds the same value as the `CREATED` column.
- `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.
- `CHARACTER_SET_CLIENT`: The session value of the `character_set_client` system variable when the routine was created. This column was added in MySQL 5.1.21.
- `COLLATION_CONNECTION`: The session value of the `collation_connection` system variable when the routine was created. This column was added in MySQL 5.1.21.
- `DATABASE_COLLATION`: The collation of the database with which the routine is associated. This column was added in MySQL 5.1.21.

Chapter 16 The INFORMATION_SCHEMA VIEWS Table

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

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		NULL
TABLE_SCHEMA		
TABLE_NAME		
VIEW_DEFINITION		
CHECK_OPTION		
IS_UPDATABLE		
DEFINER		
SECURITY_TYPE		
CHARACTER_SET_CLIENT		MySQL extension
COLLATION_CONNECTION		MySQL extension

Notes:

- The VIEW_DEFINITION column has most of what you see in the Create Table field 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
  ```

- The CHECK_OPTION column has a value of NONE, CASCADE, or LOCAL.

- 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 will be rejected. (Note that 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 has a value of DEFINER or INVOKER.

- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the view was created. This column was added in MySQL 5.1.21.

- COLLATION_CONNECTION: The session value of the collation_connection system variable when the view was created. This column was added in MySQL 5.1.21.

MySQL lets you use 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` will not affect the results from the view. However an additional consequence is that comments prior to `SELECT` are stripped from the definition by the server.

Chapter 17 The INFORMATION_SCHEMA TRIGGERS Table

The TRIGGERS table provides information about triggers. To see information about a table's triggers, you must have the TRIGGER privilege for the table (prior to MySQL 5.1.23, you must have the SUPER privilege).

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TRIGGER_CATALOG		NULL
TRIGGER_SCHEMA		
TRIGGER_NAME	Trigger	
EVENT_MANIPULATION	Event	
EVENT_OBJECT_CATALOG		NULL
EVENT_OBJECT_SCHEMA		
EVENT_OBJECT_TABLE	Table	
ACTION_ORDER		0
ACTION_CONDITION		NULL
ACTION_STATEMENT	Statement	
ACTION_ORIENTATION		ROW
ACTION_TIMING	Timing	
ACTION_REFERENCE_OLD_TABLE		NULL
ACTION_REFERENCE_NEW_TABLE		NULL
ACTION_REFERENCE_OLD_ROW		OLD
ACTION_REFERENCE_NEW_ROW		NEW
CREATED	Created	
SQL_MODE	sql_mode	MySQL extension
DEFINER	Definer	MySQL extension
CHARACTER_SET_CLIENT	character_set_client	MySQL extension
COLLATION_CONNECTION	collation_connection	MySQL extension
DATABASE_COLLATION	Database Collation	MySQL extension

Notes:

- The names in the “SHOW Name” column refer to the SHOW TRIGGERS statement, not SHOW CREATE TRIGGER. See SHOW TRIGGERS Syntax.
- TRIGGER_SCHEMA and TRIGGER_NAME: The name of the database in which the trigger occurs and the trigger name, respectively.
- 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_SCHEMA and EVENT_OBJECT_TABLE: As noted in Using Triggers, every trigger is associated with exactly one table. These columns indicate the database in which this table occurs, and the table name, respectively.

- ACTION_ORDER: The ordinal position of the trigger's action within the list of all similar triggers on the same table. This value is always 0, because it is not possible to have more than one trigger with the same EVENT_MANIPULATION and ACTION_TIMING on the same table.

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

- ACTION_ORIENTATION: Always contains the value 'ROW'.

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

- ACTION_REFERENCE_OLD_ROW and ACTION_REFERENCE_NEW_ROW: The old and new column identifiers, respectively. This means that ACTION_REFERENCE_OLD_ROW always contains the value 'OLD' and ACTION_REFERENCE_NEW_ROW always contains the value 'NEW'.

- 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 of the user who created the trigger, in '*user_name*'@'*host_name*' format. This column was added in MySQL 5.1.2.

- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the trigger was created. This column was added in MySQL 5.1.21.

- COLLATION_CONNECTION: The session value of the collation_connection system variable when the trigger was created. This column was added in MySQL 5.1.21.

- DATABASE_COLLATION: The collation of the database with which the trigger is associated. This column was added in MySQL 5.1.21.

- The following columns currently always contain NULL: TRIGGER_CATALOG, EVENT_OBJECT_CATALOG, ACTION_CONDITION, ACTION_REFERENCE_OLD_TABLE, ACTION_REFERENCE_NEW_TABLE, and CREATED.

Example, using 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: 0
          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: NULL
                  SQL_MODE:
                   DEFINER: me@localhost
      CHARACTER_SET_CLIENT: utf8
      COLLATION_CONNECTION: utf8_general_ci
        DATABASE_COLLATION: latin1_swedish_ci
```

Chapter 18 The INFORMATION_SCHEMA PLUGINS Table

The PLUGINS table provides information about server plugins.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
PLUGIN_NAME	Name	MySQL extension
PLUGIN_VERSION		MySQL extension
PLUGIN_STATUS	Status	MySQL extension
PLUGIN_TYPE	Type	MySQL extension
PLUGIN_TYPE_VERSION		MySQL extension
PLUGIN_LIBRARY	Library	MySQL extension
PLUGIN_LIBRARY_VERSION		MySQL extension
PLUGIN_AUTHOR		MySQL extension
PLUGIN_DESCRIPTION		MySQL extension
PLUGIN_LICENSE		MySQL extension

Notes:

- The PLUGINS table is a nonstandard table. It was added in MySQL 5.1.5.
- PLUGIN_NAME is the name used to refer to the plugin in statements such as INSTALL PLUGIN and UNINSTALL PLUGIN.
- PLUGIN_VERSION is the version from the plugin's general type descriptor.
- PLUGIN_STATUS indicates the plugin status, one of ACTIVE, INACTIVE, DISABLED, or DELETED.
- PLUGIN_TYPE indicates the type of plugin, such as STORAGE ENGINE or INFORMATION_SCHEMA.
- PLUGIN_TYPE_VERSION is the version from the plugin's type-specific descriptor.
- PLUGIN_LIBRARY is the name of the plugin shared object 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 indicates the plugin API interface version.
- PLUGIN_AUTHOR names the plugin author.
- PLUGIN_DESCRIPTION provides a short description of the plugin.
- PLUGIN_LICENSE indicates how the plugin is licensed; for example, GPL. This column was added in MySQL 5.1.12.

For plugins installed with INSTALL PLUGIN, the PLUGIN_NAME and PLUGIN_LIBRARY values are also registered in the mysql.plugin table.

These statements are equivalent:

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

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 using the `SHOW PLUGINS` statement. See SHOW PLUGINS Syntax.

Chapter 19 The INFORMATION_SCHEMA ENGINES Table

The ENGINES table provides information about storage engines.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
ENGINE	Engine	MySQL extension
SUPPORT	Support	MySQL extension
COMMENT	Comment	MySQL extension
TRANSACTIONS	Transactions	MySQL extension
XA	XA	MySQL extension
SAVEPOINTS	Savepoints	MySQL extension

Notes:

- The ENGINES table is a nonstandard table. It was added in MySQL 5.1.5. Its contents correspond to the columns of the SHOW ENGINES statement. For descriptions of its columns, see SHOW ENGINES Syntax.

See also SHOW ENGINES Syntax.

Chapter 20 The INFORMATION_SCHEMA PARTITIONS Table

The PARTITIONS table provides information about table partitions. See Partitioning, for more information about partitioning tables.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
TABLE_CATALOG		MySQL extension
TABLE_SCHEMA		MySQL extension
TABLE_NAME		MySQL extension
PARTITION_NAME		MySQL extension
SUBPARTITION_NAME		MySQL extension
PARTITION_ORDINAL_POSITION		MySQL extension
SUBPARTITION_ORDINAL_POSITION		MySQL extension
PARTITION_METHOD		MySQL extension
SUBPARTITION_METHOD		MySQL extension
PARTITION_EXPRESSION		MySQL extension
SUBPARTITION_EXPRESSION		MySQL extension
PARTITION_DESCRIPTION		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
PARTITION_COMMENT		MySQL extension
NODEGROUP		MySQL extension
TABLESPACE_NAME		MySQL extension

Notes:

- The PARTITIONS table is a nonstandard table. It was added in MySQL 5.1.6.

  Each record in this table corresponds to an individual partition or subpartition of a partitioned table.

- TABLE_CATALOG: This column is always NULL.
- TABLE_SCHEMA: This column contains the name of the database to which the table belongs.
- TABLE_NAME: This column contains the name of the table containing the partition.
- PARTITION_NAME: The name of the partition.

- SUBPARTITION_NAME: If the PARTITIONS table record represents a subpartition, then this column contains the name of subpartition; otherwise it is 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: This is 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 record 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              |
  +----------------------+
  1 row in set (0.09 sec)
  ```

- 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, then 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 comma-separated list of 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.

Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, TABLE_ROWS shows correct information for NDB tables. Previously, for partitions of NDB tables, the TABLE_ROWS column value was always 0.

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.

  Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, AVG_ROW_LENGTH includes statistics for partitions of NDB tables, whether the tables use implicit or explicit partitioning. (Previously the value of this column was always 0 for partitions of NDB tables.)

  You can also obtain equivalent 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.

  Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, DATA_LENGTH shows correct information for in-memory data in NDB tables. Previously, for partitions of NDB tables, the DATA_LENGTH column value was always 0.

  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.

  Beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, MAX_DATA_LENGTH shows the space allocated for the disk part of a MySQL Cluster Disk Data table or fragment. (Previously, for partitions of NDB tables, the MAX_DATA_LENGTH column value was always NULL.)

  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 MySQL Cluster Disk Data tables, beginning with MySQL Cluster NDB 7.0.22 and MySQL Cluster NDB 7.1.11, DATA_FREE shows the space allocated on disk for, but not used by, a Disk Data table or fragment on disk. (Previously, for partitions of NDB tables, the value of this column was always 0.)

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

- CREATE_TIME: The time of the partition's or subpartition's creation.

  For partitioned InnoDB tables, this column is always NULL.

- UPDATE_TIME: The time that the partition or subpartition was last modified.

  For partitioned InnoDB tables, this column is always NULL.

- CHECK_TIME: The last time that the table to which this partition or subpartition belongs was checked.

For partitioned InnoDB tables, this column is always NULL.

- CHECKSUM: The checksum value, if any; otherwise, this column is NULL.
- PARTITION_COMMENT: This column contains the text of any comment made for the partition.

  In MySQL 5.1, the display width of this column is 80 characters, and partition comments which exceed this length are truncated to fit. This issue is fixed in MySQL 5.6. (Bug #11748924, Bug #37728)

  The default value for this column is an empty string.
- NODEGROUP: This is the nodegroup to which the partition belongs. This is relevant only to MySQL Cluster tables; otherwise the value of this column is always 0.
- TABLESPACE_NAME: This column contains the name of the tablespace to which the partition belongs. The value of this column is always DEFAULT.
- **Important**

  If any partitioned tables created in a MySQL version prior to MySQL 5.1.6 are present following an upgrade to MySQL 5.1.6 or later, it is not possible to SELECT from, SHOW, or DESCRIBE the PARTITIONS table. See the Release Notes for MySQL 5.1.6 *before* upgrading from MySQL 5.1.5 or earlier to MySQL 5.1.6 or later.
- A nonpartitioned table has one record in INFORMATION_SCHEMA.PARTITIONS; 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. (The PARTITION_COMMENT column in this case is blank.)

  In MySQL 5.1, there is also only one record in the PARTITIONS table for a table using the NDBCLUSTER storage engine. The same columns are also NULL (or empty) as for a nonpartitioned table.

Chapter 21 The INFORMATION_SCHEMA EVENTS Table

The EVENTS table provides information about scheduled events, which are discussed in Using the Event Scheduler. The SHOW Name values correspond to column names of the SHOW EVENTS statement.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
EVENT_CATALOG		NULL, MySQL extension
EVENT_SCHEMA	Db	MySQL extension
EVENT_NAME	Name	MySQL extension
DEFINER	Definer	MySQL extension
TIME_ZONE	Time zone	MySQL extension
EVENT_BODY		MySQL extension
EVENT_DEFINITION		MySQL extension
EVENT_TYPE	Type	MySQL extension
EXECUTE_AT	Execute at	MySQL extension
INTERVAL_VALUE	Interval value	MySQL extension
INTERVAL_FIELD	Interval field	MySQL extension
SQL_MODE		MySQL extension
STARTS	Starts	MySQL extension
ENDS	Ends	MySQL extension
STATUS	Status	MySQL extension
ON_COMPLETION		MySQL extension
CREATED		MySQL extension
LAST_ALTERED		MySQL extension
LAST_EXECUTED		MySQL extension
EVENT_COMMENT		MySQL extension
ORIGINATOR	Originator	MySQL extension
CHARACTER_SET_CLIENT	character_set_client	MySQL extension
COLLATION_CONNECTION	collation_connection	MySQL extension
DATABASE_COLLATION	Database Collation	MySQL extension

Notes:

- The EVENTS table is a nonstandard table. It was added in MySQL 5.1.6.
- EVENT_CATALOG: The value of this column is always NULL.
- EVENT_SCHEMA: The name of the schema (database) to which this event belongs.
- EVENT_NAME: The name of the event.
- DEFINER: The account of 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.

This column was added in MySQL 5.1.17. See Release Notes for MySQL 5.1.17 for important information if you are using the Event Scheduler and are upgrading to MySQL 5.1.17 or later from an earlier version.

- `EVENT_BODY`: The language used for the statements in the event's `DO` clause; in MySQL 5.1, this is always `SQL`.

  This column was added in MySQL 5.1.12. It is not to be confused with the column of the same name (now named `EVENT_DEFINITION`) that existed in earlier MySQL versions.

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

  > **Note**
  >
  > Prior to MySQL 5.1.12, this column was named `EVENT_BODY`.

- `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 2006-02-09 14:05:30, the value shown in this column would be `'2006-02-10 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 recurring events, this column contains the numeric portion of the event's `EVERY` clause.

  For a one-time event (that is, an event whose timing is determined by an `AT` clause), this column is `NULL`.

- `INTERVAL_FIELD`: For recurring events, this column contains the units portion of the `EVERY` clause governing the timing of the event. Thus, this column contains a value such as '`YEAR`', '`QUARTER`', '`DAY`', and so on.

  > **Note**
  >
  > In early MySQL 5.1 releases, this value was prefixed with '`INTERVAL_`', and was displayed as '`INTERVAL_YEAR`', '`INTERVAL_QUARTER`', '`INTERVAL_DAY`', and so on.

  For a one-time event (that is, an event whose timing is determined by an `AT` clause), this column is `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`: 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`. (Prior to MySQL 5.1.8, it contained `0000-00-00 00:00:00` in such cases.)

- 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: One of the three values ENABLED, DISABLED, or SLAVESIDE_DISABLED.

  SLAVESIDE_DISABLED was added to the list of possible values for this column in MySQL 5.1.18. This value indicates that the creation of the event occurred on another MySQL server acting as a replication master and was replicated to the current MySQL server which is acting as a slave, but the event is not presently being executed on the slave. See Replication of Invoked Features, for more 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 column holds the same value as the CREATED column.

- LAST_EXECUTED: The date and time when the event last executed. A DATETIME value. If the event has never executed, this column is NULL.

  Before MySQL 5.1.23, LAST_EXECUTED indicates when event finished executing. As of 5.1.23, LAST_EXECUTED instead indicates when the event started. As a result, the ENDS column is never less than LAST_EXECUTED.

- EVENT_COMMENT: The text of a comment, if the event has one. If not, the value of this column is an empty string.

- ORIGINATOR: The server ID of the MySQL server on which the event was created; used in replication. The default value is 0. This column was added in MySQL 5.1.18.

- CHARACTER_SET_CLIENT: The session value of the character_set_client system variable when the event was created. This column was added in MySQL 5.1.21.

- COLLATION_CONNECTION: The session value of the collation_connection system variable when the event was created. This column was added in MySQL 5.1.21.

- DATABASE_COLLATION: The collation of the database with which the event is associated. This column was added in MySQL 5.1.21.

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: NULL
        EVENT_SCHEMA: test
          EVENT_NAME: e_daily
             DEFINER: me@localhost
           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:
              STARTS: 2008-09-03 12:13:39
                ENDS: NULL
              STATUS: ENABLED
       ON_COMPLETION: NOT PRESERVE
             CREATED: 2008-09-03 12:13:39
        LAST_ALTERED: 2008-09-03 12:13:39
       LAST_EXECUTED: NULL
       EVENT_COMMENT: Saves total number of sessions then clears the
                      table each day
          ORIGINATOR: 1
CHARACTER_SET_CLIENT: latin1
COLLATION_CONNECTION: latin1_swedish_ci
  DATABASE_COLLATION: latin1_swedish_ci
```

Times in the EVENTS table are displayed using the event time zone or the current session time zone. Prior to MySQL 5.1.17, some of the times are displayed in UTC rather than the event time zone. For details, see Event Metadata.

See also SHOW EVENTS Syntax.

Chapter 22 The INFORMATION_SCHEMA FILES Table

The FILES table provides information about the files in which MySQL NDB Disk Data tables are stored.

Note

This table provides information about Disk Data *files* only; you cannot use it for determining disk space allocation or availability for individual NDB tables. However, beginning with MySQL Cluster NDB 6.3.27 and MySQL Cluster NDB 7.0.8, it is possible to see how much space is allocated for each NDB table having data stored on disk, as well as how much remains available for storage of data on disk for that table, using ndb_desc. For more information, see ndb_desc — Describe NDB Tables.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
FILE_ID		MySQL extension
FILE_NAME		MySQL extension
FILE_TYPE		MySQL extension
TABLESPACE_NAME		MySQL extension
TABLE_CATALOG		MySQL extension
TABLE_SCHEMA		MySQL extension
TABLE_NAME		MySQL extension
LOGFILE_GROUP_NAME		MySQL extension
LOGFILE_GROUP_NUMBER		MySQL extension
ENGINE		MySQL extension
FULLTEXT_KEYS		MySQL extension
DELETED_ROWS		MySQL extension
UPDATE_COUNT		MySQL extension
FREE_EXTENTS		MySQL extension
TOTAL_EXTENTS		MySQL extension
EXTENT_SIZE		MySQL extension
INITIAL_SIZE		MySQL extension
MAXIMUM_SIZE		MySQL extension
AUTOEXTEND_SIZE		MySQL extension
CREATION_TIME		MySQL extension
LAST_UPDATE_TIME		MySQL extension
LAST_ACCESS_TIME		MySQL extension
RECOVER_TIME		MySQL extension
TRANSACTION_COUNTER		MySQL extension
VERSION		MySQL extension
ROW_FORMAT		MySQL extension
TABLE_ROWS		MySQL extension
AVG_ROW_LENGTH		MySQL extension

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
DATA_LENGTH		MySQL extension
MAX_DATA_LENGTH		MySQL extension
INDEX_LENGTH		MySQL extension
DATA_FREE		MySQL extension
CREATE_TIME		MySQL extension
UPDATE_TIME		MySQL extension
CHECK_TIME		MySQL extension
CHECKSUM		MySQL extension
STATUS		MySQL extension
EXTRA		MySQL extension

Notes:

- FILE_ID column values are auto-generated.
- FILE_NAME is the name of an UNDO log file created by CREATE LOGFILE GROUP or ALTER LOGFILE GROUP, or of a data file created by CREATE TABLESPACE or ALTER TABLESPACE.
- FILE_TYPE is one of the values UNDOFILE or DATAFILE.

  Beginning with MySQL Cluster NDB 6.2.19, MySQL Cluster NDB 6.3.32, MySQL Cluster NDB 7.0.13, and MySQL Cluster NDB 7.1.2, this column can also have the value TABLESPACE.
- TABLESPACE_NAME is the name of the tablespace with which the file is associated.
- The value of the TABLESPACE_CATALOG column is always NULL.
- TABLE_NAME is the name of the Disk Data table with which the file is associated, if any.
- The LOGFILE_GROUP_NAME column gives the name of the log file group to which the log file or data file belongs.
- For an UNDO log file, the LOGFILE_GROUP_NUMBER contains the auto-generated ID number of the log file group to which the log file belongs.
- For a MySQL Cluster Disk Data log file or data file, the value of the ENGINE column is always NDB or NDBCLUSTER.
- For a MySQL Cluster Disk Data log file or data file, the value of the FULLTEXT_KEYS column is always empty.
- The FREE EXTENTS column displays the number of extents which have not yet been used by the file. The TOTAL EXTENTS column show the total number of extents allocated to the file.

  The difference between these two 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';
  ```

  You can approximate the amount of disk space in use by the file by multiplying this 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. See CREATE TABLESPACE Syntax, for more information.

- The INITIAL_SIZE column shows the size in bytes of the file. 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.

  For MySQL Cluster Disk Data files, the value of the MAXIMUM_SIZE column is always the same as INITIAL_SIZE, and the AUTOEXTEND_SIZE column is always empty.

- The CREATION_TIME column shows the date and time when the file was created. The LAST_UPDATE_TIME column displays the date and time when the file was last modified. The LAST_ACCESSED column provides the date and time when the file was last accessed by the server.

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

- For MySQL Cluster Disk Data files, the value of the RECOVER_TIME and TRANSACTION_COUNTER columns is always 0.

- For MySQL Cluster Disk Data files, the following columns are always NULL:

  - VERSION
  - ROW_FORMAT
  - TABLE_ROWS
  - AVG_ROW_LENGTH
  - DATA_LENGTH
  - MAX_DATA_LENGTH

- INDEX_LENGTH
- DATA_FREE
- CREATE_TIME
- UPDATE_TIME
- CHECK_TIME
- CHECKSUM

- For MySQL Cluster Disk Data files, the value of the STATUS column is always NORMAL.
- For MySQL Cluster Disk Data files, the EXTRA column shows which data node the file belongs to, as each data node has its own copy of the file. Suppose that you use this statement on a MySQL Cluster with four data nodes:

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

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

```
mysql> SELECT LOGFILE_GROUP_NAME, FILE_TYPE, EXTRA
    ->     FROM INFORMATION_SCHEMA.FILES
    ->     WHERE FILE_NAME = 'new_undo.dat';
+--------------------+-------------+----------------+
| LOGFILE_GROUP_NAME | FILE_TYPE   | EXTRA          |
+--------------------+-------------+----------------+
| mygroup            | UNDO FILE   | CLUSTER_NODE=3 |
| mygroup            | UNDO FILE   | CLUSTER_NODE=4 |
| mygroup            | UNDO FILE   | CLUSTER_NODE=5 |
| mygroup            | UNDO FILE   | CLUSTER_NODE=6 |
+--------------------+-------------+----------------+
4 rows in set (0.01 sec)
```

- The FILES table is a nonstandard table. It was added in MySQL 5.1.6.
- Beginning with MySQL 5.1.14, an additional row is present in the FILES table following the creation of a logfile group. This row has NULL for the value of the FILE_NAME column. For this row, the value of the FILE_ID column is always 0, that of the FILE_TYPE column is always UNDO FILE, and that of the STATUS column is always NORMAL. The value of the ENGINE column is always NDBCLUSTER.

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

  Suppose there are no existing log file groups on your MySQL 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;
```

```
Query OK, 0 rows affected (3.81 sec)
```

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 |
+--------------+---------+---------+------+----------+
2 rows in set (0.01 sec)
```

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;
Query OK, 0 rows affected (1.02 sec)
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 |
+----------------+---------+---------+------+----------+
3 rows in set (0.01 sec)
```

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 |
+--------------+------------+
1 row in set (0.02 sec)
```

If you create a MySQL 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;
Query OK, 0 rows affected (8.71 sec)
mysql> CREATE TABLE dd (
    ->   c1 INT NOT NULL PRIMARY KEY,
    ->   c2 INT,
    ->   c3 DATE
    ->   )
    ->   TABLESPACE ts1 STORAGE DISK
    ->   ENGINE = NDB;
Query OK, 0 rows affected (2.11 sec)
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');
Query OK, 4 rows affected (0.01 sec)
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 |
+--------------+------------+
1 row in set (0.01 sec)
```

- Beginning with MySQL Cluster NDB 6.2.19, MySQL Cluster NDB 6.3.32, MySQL Cluster NDB 7.0.13, and MySQL Cluster NDB 7.1.2, an additional row is present in the FILES table for any tablespace, whether or not any data files are associated with the tablespace. This row has NULL for the value of the FILE_NAME column. For this row, the value of the FILE_ID column is always 0, that of the FILE_TYPE column is always TABLESPACE, and that of the STATUS column is always NORMAL. The value of the ENGINE column is always NDBCLUSTER.
- There are no SHOW statements associated with the FILES table.
- For additional information, and examples of creating and dropping MySQL Cluster Disk Data objects, see MySQL Cluster Disk Data Tables.

Chapter 23 The INFORMATION_SCHEMA PROCESSLIST Table

The PROCESSLIST table provides information about which threads are running.

INFORMATION_SCHEMA Name	SHOW Name	Remarks
ID	Id	MySQL extension
USER	User	MySQL extension
HOST	Host	MySQL extension
DB	db	MySQL extension
COMMAND	Command	MySQL extension
TIME	Time	MySQL extension
STATE	State	MySQL extension
INFO	Info	MySQL extension

For an extensive description of the table columns, see SHOW PROCESSLIST Syntax.

Notes:

- The PROCESSLIST table is a nonstandard table. It was added in MySQL 5.1.7.
- Like the output from the corresponding SHOW statement, the PROCESSLIST table will only show information about your own threads, unless you have the PROCESS privilege, in which case you will see information about other threads, too. As an anonymous user, you cannot see any rows at all.
- If an SQL statement refers to INFORMATION_SCHEMA.PROCESSLIST, 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, though.

The following statements are equivalent:

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

Chapter 24 The INFORMATION_SCHEMA REFERENTIAL_CONSTRAINTS Table

The REFERENTIAL_CONSTRAINTS table provides information about foreign keys.

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
CONSTRAINT_CATALOG		NULL
CONSTRAINT_SCHEMA		
CONSTRAINT_NAME		
UNIQUE_CONSTRAINT_CATALOG		NULL
UNIQUE_CONSTRAINT_SCHEMA		
UNIQUE_CONSTRAINT_NAME		
MATCH_OPTION		
UPDATE_RULE		
DELETE_RULE		
TABLE_NAME		
REFERENCED_TABLE_NAME		

Notes:

- The REFERENTIAL_CONSTRAINTS table was added in MySQL 5.1.10. The REFERENCED_TABLE_NAME column was added in MySQL 5.1.16.
- TABLE_NAME has the same value as TABLE_NAME in INFORMATION_SCHEMA.TABLE_CONSTRAINTS.
- CONSTRAINT_SCHEMA and CONSTRAINT_NAME identify the foreign key.
- UNIQUE_CONSTRAINT_SCHEMA, UNIQUE_CONSTRAINT_NAME, and REFERENCED_TABLE_NAME identify the referenced key. (Note: Before MySQL 5.1.16, UNIQUE_CONSTRAINT_NAME incorrectly named the referenced table, not the constraint.)
- The only valid value at this time for MATCH_OPTION is NONE.
- The possible values for UPDATE_RULE or DELETE_RULE are CASCADE, SET NULL, SET DEFAULT, RESTRICT, NO ACTION.

Chapter 25 The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables

The `GLOBAL_STATUS` and `SESSION_STATUS` tables provide information about server status variables. Their contents correspond to the information produced by the `SHOW GLOBAL STATUS` and `SHOW SESSION STATUS` statements (see SHOW STATUS Syntax).

`INFORMATION_SCHEMA` **Name**	**`SHOW`** **Name**	**Remarks**
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

Notes:

- The `GLOBAL_STATUS` and `SESSION_STATUS` tables were added in MySQL 5.1.12.
- The `VARIABLE_VALUE` column for each of these tables is defined as `VARCHAR(1024)`. Previously, this column had the data type `DECIMAL(22,7)`, but was changed to `VARCHAR(20480)` in 5.1.19 to avoid loss of data when working with status variables whose values were strings, and shortened to `VARCHAR(1024)` in 5.1.31 to improve performance.

Chapter 26 The INFORMATION_SCHEMA GLOBAL_VARIABLES and SESSION_VARIABLES Tables

The GLOBAL_VARIABLES and SESSION_VARIABLES tables provide information about server status variables. Their contents correspond to the information produced by the SHOW GLOBAL VARIABLES and SHOW SESSION VARIABLES statements (see SHOW VARIABLES Syntax).

INFORMATION_SCHEMA Name	**SHOW Name**	**Remarks**
VARIABLE_NAME	Variable_name	
VARIABLE_VALUE	Value	

Notes:

- The GLOBAL_VARIABLES and SESSION_VARIABLES tables were added in MySQL 5.1.12.

- The VARIABLE_VALUE column for each of these tables is defined as VARCHAR(1024). Previously, this column had the data type LONGTEXT; this was changed in MySQL 5.1.19 to VARCHAR(20480) and in 5.1.31 to VARCHAR(1024) to make these tables consistent with the GLOBAL_STATUS and SESSION_STATUS tables, whose definitions were changed in those versions (see Chapter 25, *The INFORMATION_SCHEMA GLOBAL_STATUS and SESSION_STATUS Tables*).

  For variables with very long values that are not completely displayed, use SELECT as a workaround. For example:

  ```
  SELECT @@GLOBAL.innodb_data_file_path;
  ```

Chapter 27 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                                 |
| GLOBAL_STATUS                         |
| GLOBAL_VARIABLES                      |
| KEY_COLUMN_USAGE                      |
| PARTITIONS                            |
| PLUGINS                               |
| PROCESSLIST                           |
| REFERENTIAL_CONSTRAINTS               |
| ROUTINES                              |
| SCHEMATA                              |
| SCHEMA_PRIVILEGES                     |
| SESSION_STATUS                        |
| SESSION_VARIABLES                     |
| STATISTICS                            |
| TABLES                                |
| TABLE_CONSTRAINTS                     |
| TABLE_PRIVILEGES                      |
| TRIGGERS                              |
| USER_PRIVILEGES                       |
| VIEWS                                 |
+---------------------------------------+
27 rows in set (0.00 sec)
```

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 28 MySQL 5.1 FAQ: INFORMATION_SCHEMA

Questions

- 28.1: Where can I find documentation for the MySQL `INFORMATION_SCHEMA` database?
- 28.2: Is there a discussion forum for `INFORMATION_SCHEMA`?
- 28.3: Where can I find the ANSI SQL 2003 specification for `INFORMATION_SCHEMA`?
- 28.4: What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?
- 28.5: Can I add to or otherwise modify the tables found in the `INFORMATION_SCHEMA` database?

Questions and Answers

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

See Chapter 1, *INFORMATION_SCHEMA Tables*

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

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

28.3: 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—which give a comprehensive overview of the standard, including `INFORMATION_SCHEMA`.

28.4: What is the difference between the Oracle Data Dictionary and MySQL's `INFORMATION_SCHEMA`?

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

28.5: 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*.